

BOOK LIBRARY
POSTGRADUATE SCHOOL
MAY, CALIF. 93940

NAVAL POSTGRADUATE SCHOOL

Monterey, California



THESIS

A VERSION OF THE GRAPHICS-ORIENTED
INTERACTIVE FINITE ELEMENT TIME-SHARING
SYSTEM (GIFTS) FOR AN IBM WITH CP/CMS

by

Ronnie Hundley

March 1982

Thesis Advisor:

G. Cantin

Approved for public release; distribution unlimited.

T204040

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) A Version of the Graphics-oriented Inter- active Finite Element Time-sharing System (GIFTS) for an IBM with CP/CMS		5. TYPE OF REPORT & PERIOD COVERED Master's & Engineer's Thesis - March 1982
7. AUTHOR(s) Ronnie Hundley		6. PERFORMING ORG. REPORT NUMBER
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		8. CONTRACT OR GRANT NUMBER(s)
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office)		12. REPORT DATE March 1982
		13. NUMBER OF PAGES 96
		15. SECURITY CLASS. (of this report) Unclassified
		15a. DECLASSIFICATION/DOWNGRADING SCHEDULE
16. DISTRIBUTION STATEMENT (of this Report) Approved for public release; distribution unlimited.		
17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)		
18. SUPPLEMENTARY NOTES		
19. KEY WORDS (Continue on reverse side if necessary and identify by block number) GIFTS Finite Element Structural Analysis Computer Graphics Tektronix IBM Computer		
20. ABSTRACT (Continue on reverse side if necessary and identify by block number) A version of the Graphics-oriented, Interactive, Finite element, Time-sharing System (GIFTS) has been developed for, and installed on, an IBM computer with the Conversational Monitor System (CMS). GIFTS, developed at, and available from the Interactive Graphics Engineering Laboratory of the University of Arizona, is an extensive code for static, tran- sient, modal, and constrained substructural analysis of three		

dimensional truss, plate, shell, and solid finite element models. A brief description of GIFTS, including insights into its logic and structure necessary to the version's development, and an in-depth description of the method used to invoke CMS commands from the executing program for the purpose of data base management are provided. The version, making use of the Tektronix 4000 series graphics terminals, is self-contained and portable, allowing its installation on other IBM computers with the CMS operating system.

Approved for public release; distribution unlimited.

A Version of the Graphics-oriented Interactive
Finite Element Time-sharing System (GIFTS)
for an IBM with CP/CMS

by

Ronnie Hundley
Lieutenant, United States Navy
B.S., University of Washington, 1974

Submitted in partial fulfillment of the
requirements for the degrees of

MASTER OF SCIENCE IN MECHANICAL ENGINEERING

and

MECHANICAL ENGINEER

from the

NAVAL POSTGRADUATE SCHOOL
March 1982

ABSTRACT

A version of the Graphics-oriented, Interactive, Finite element, Time-sharing System (GIFTS) has been developed for, and installed on, an IBM computer with the Conversational Monitor System (CMS). GIFTS, developed at, and available from the Interactive Graphics Engineering Laboratory of the University of Arizona, is an extensive code for static, transient, modal, and constrained substructural analysis of three dimensional truss, plate, shell, and solid finite element models. A brief description of GIFTS, including insights into its logic and structure necessary to the version's development, and an in-depth description of the method used to invoke CMS commands from the executing program for the purpose of data base management are provided. The version, making use of the Tektronix 4000 series graphics terminals, is self-contained and portable, allowing its installation on other IBM computers with the CMS operating system.

TABLE OF CONTENTS

I.	INTRODUCTION-----	8
II.	A BRIEF DESCRIPTION OF GIFTS-----	10
	A. THE MODULES-----	11
	B. THE LIBRARIES-----	11
	1. The Graphics Package-----	12
	2. The Character Manipulation Package-----	12
	3. The Data Base Management Package-----	12
III.	IBM IMPLEMENTATION-----	13
	A. THE SYSTEM-----	13
	1. The Control Program (CP)-----	14
	2. The Conversational Monitor System (CMS)-----	14
	a. Invoking CMS Commands from an Executing Program-----	14
	b. The CMS File Identifier-----	15
	B. THE IBM (CP/CMS) GRAPHICS PACKAGE-----	16
	1. ASCII/EBCDIC Translation-----	16
	2. Flushing the Graphics Buffer to the Terminal-----	18
	C. THE IBM (CP/CMS) CHARACTER MANIPULATION PACKAGE-----	20
	D. THE IBM (CP/CMS) DATA BASE MANAGEMENT PACKAGE-----	21
	1. The IBM (CP/CMS) Data Base-----	21
	a. The Direct (Random) Access Files-----	22
	b. Sequential Data Base Files-----	23
	c. Special Sequential Data Base Files-----	24

2.	Defining the Direct Access Files and GIFTS Module IBMUDB-----	26
3.	Opening of the Data Base Files-----	31
4.	Closing Files-----	34
5.	Deleting Files-----	36
6.	Renaming Files-----	37
7.	Checking for a File's Presence-----	39
E.	THE ADDED ASSEMBLY ROUTINES (LIBR5A)-----	40
F.	IBM (CP/CMS) INSTALLATION INSTRUCTIONS-----	40
G.	IBM (CP/CMS) USER INSTRUCTIONS-----	42
IV.	SUMMARY AND RECOMMENDATIONS-----	43
APPENDIX A:	DESCRIPTION OF GIFTS MODULES-----	44
APPENDIX B:	SUMMARY OF CMS COMMANDS USED-----	49
APPENDIX C:	THE UNIFIED DATA BASE FOR THE IBM-----	51
APPENDIX D:	LISTING OF NEW GIFTS PROGRAM MODULE IBMUDB---	55
APPENDIX E:	THE IBM (CP/CMS) GRAPHICS PACKAGE SUBROUTINES-----	58
APPENDIX F:	THE IBM (CP/CMS) CHARACTER MANIPULATION PACKAGE SUBROUTINES-----	62
APPENDIX G:	THE IBM (CP/CMS) DATA BASE MANAGEMENT PACKAGE SUBROUTINES-----	66
APPENDIX H:	LISTING OF THE IBM (CP/CMS) ASSEMBLY ROUTINES-----	70
APPENDIX I:	SOME IBM INSTALLATION TOOLS-----	88
APPENDIX J:	GIFTS USER INSTRUCTIONS FOR THE IBM (CP/CMS) VERSION-----	92
	LIST OF REFERENCES-----	95
	INITIAL DISTRIBUTION LIST-----	96

ACKNOWLEDGEMENT

I extend to Professor Gilles Cantin my most sincere thanks for guiding me through this growing process, and for placing me upon the threshold of the rapidly expanding opportunities of computer aided design.

To my wonderfully understanding wife, Kathy, and my ever changing children, Noel, Rhonda, and Maria, I extend my love and devotion and promises of compensation for the sacrifices made by them during my studies.

I. INTRODUCTION

The purpose of this thesis is to describe the development of an IBM (CP/CMS)¹ version of the Graphics-oriented Interactive Finite Element Time-sharing System (GIFTS). GIFTS, developed at, and available from the University of Arizona, is a set of computer programs designed to perform static and dynamic structural finite element analysis, making extensive use of interactive computer graphics which allows the user to ensure correct structural modeling and to view the results of the analysis in a manner most understandable to him.

Versions of GIFTS have been developed for several different computers but not for the IBM, and in particular, for an IBM with CP/CMS. At the Naval Postgraduate School it was desired to provide GIFTS as an instructional tool to the students on a system for which they enjoy ready access and were familiar. This system is the school's main frame, an IBM 3033 with CP/CMS. It was to meet this need that the development was undertaken. A self-contained IBM (CP/CMS) version of GIFTS for an IBM with CP/CMS was developed in such a way as to allow easy implementation on other IBM systems with CP/CMS.

¹International Business Machine with the Command Program and Conversational Monitor System.

It is not the intent of this thesis to provide a detailed description of GIFTS, for this, the reader should consult the GIFTS reference manuals [Refs. 1-4]. It is the intent to provide information on those areas of GIFTS logic and structure that are necessary to the understanding of this version's development and to provide some guidance in the implementation and use of the GIFTS IBM (CP/CMS) version.

II. A BRIEF DESCRIPTION OF GIFTS

GIFTS, developed by Professor Hussein A. Kamel and Mr. Michael W. McCabe of the University of Arizona, is an interactive finite element analysis system designed to provide the user with a unified approach to model generation, model display (tabular or graphical), analysis and result display, with a minimum of input by the user. It may be used as a self-contained system, or as a pre- and post-processor for other systems.² It is operational on many systems, including mini-computers with cores having as few as 32,000 words of addressable memory (on mini-computers, the program modules must be overlayed).

GIFTS consists of over 100,000 lines of FORTRAN source code divided among 25 individually executable program modules. The modules, being run independently of each other, communicate by means of an automatically managed, disk resident data base, known as the Unified Data Base (UDB). To perform a complete finite element analysis with GIFTS, the user executes a GIFTS procedure, using a number of modules in a specified order depending on the type of analysis being performed. GIFTS has procedures for static, transient, modal, and constrained substructural analysis of three

²Interfaces for pre- and post-processing are available for ANSYS, SAP4, and NASTRAN.

dimensional truss, plate, shell, and solid finite element models. The program modules use interactive computer graphics extensively for viewing the results of model generation and problem solution. GIFTS can also provide the user with the information in tabular form.

A. THE MODULES

The GIFTS program modules vary in size from a few hundred lines to over 12,000 lines of machine independent code. They can be grouped according to their intended use. These groupings are:

- 1) the model generation and editing modules,
- 2) the load and boundary condition generation, display and editing modules,
- 3) the general purpose computational and result display modules,
- 4) the natural vibration analysis modules,
- 5) the transient analysis modules, and
- 6) the constrained substructuring modules.

A brief description of the program modules can be found in Appendix A.

B. THE LIBRARIES

Commonly used subroutines and functions are grouped into a user library called the GIFTS library. For convenience, the GIFTS library is subdivided into five groups called LIBR1, LIBR2, LIBR3, LIBR4, and LIBR5. Of these groups, only LIBR5 contains machine dependent subroutines and

functions. LIBR5 is composed of approximately 30 routines containing machine dependencies that must be modified for each computer system on which GIFTS is implemented. The routines in LIBR5 are grouped according to function. The groups are the Graphics Package, the Character Manipulation Package, and the Data Base Management Package. It was through the modification of, and addition to, these packages that the IBM (CP/CMS) version was developed.

1. The Graphics Package

This is a set of subroutines containing all the necessary software to drive a Tektronix 4000 series graphics terminal. If a different terminal is desired for the graphics output, these subroutines are the ones to be modified.

2. The Character Manipulation Package

These subroutines are used to pack, unpack, compare, and change the format of characters and character strings. They conduct interactive I/O with the user and make Operating System calls for the date, time of day, and CPU time used.

3. The Data Base Management Package

This package of routines performs data base management by invoking primitive file handling functions, such as opening, closing, defining, extending, renaming, printing, and deleting UDB files. It additionally performs all I/O operations with the direct access UDB files.

III. IBM IMPLEMENTATION

Three requirements were established for this IBM (CP/CMS) version development of GIFTS. They were: to maintain the logic of the machine independent routines³ (that is, not to alter them); to continue the use of the Tektronix 4000 series terminals for graphics; and to provide a self-contained package of routines that would allow installation on other IBM systems with little or no alteration. The only requirement not met was the first, to maintain the logic of the machine independent routines, but it was only necessary to modify subroutine FRESLT⁴ in LIBR1. This alteration and the accomplishment of the other requirements are discussed below.

A. THE SYSTEM

The IBM (CP/CMS) version was developed on an IBM 3033 with the Virtual Machine Facility/370 (VM/370) System. It should be noted that VM is not the operating system but rather the virtual machine 'manager'.

The VM/370 system supervises four components: the control program (CP), the conversational monitor system (CMS), the remote spooling communications subsystem (RSCS), and

³The GIFTS' program modules, LIBR1, LIBR2, LIBR3, and LIBR4.

⁴See the Section on the Opening of Files in this Chapter.

the interactive problem control system (IPCS). Only two of these components, CP and CMS, were necessary for the implementation of GIFTS. More detailed information on CP/CMS can be obtained from References [5] through [8].

1. The Control Program (CP)

CP allocates to the virtual machine of each user the resources necessary for proper interface between the virtual machine, with associated virtual input/output (I/O) devices, and the real computer with associated real I/O devices. It additionally allows communication between virtual machines,

2. The Conversational Monitor System (CMS)

CMS is the operating system of the virtual machine and as such has commands for editing, compiling, loading (linking), and executing programs. It also has commands for file manipulation that can be invoked from the CMS environment, an 'EXEC' file⁵, or from an executing program. It is through the use of these file manipulation commands, invoked from GIFTS during execution, that the development of this version was made possible. A brief description of the CMS commands used in the IBM (CP/CMS) version can be found in Appendix B.

- a. Invoking CMS Commands from an Executing Program

In order to invoke a CMS command from an executing program it was necessary to provide an assembly language

⁵This is the same as a command file in other operating systems.

program that would execute CMS commands passed to it from the executing program. To this end, the assembly program FRTCMX was developed. A listing of FRTCMX can be found in the listing of LIBR5A in Appendix H. FRTCMX executes the CMS commands in the CMS environment, and then returns control to the calling FORTRAN program. As an example, the FORTRAN statement:

```
CALL FRTCMX (IERR,'ERASE  ','PIPJOINT','PTS  ')
```

would delete the file 'PIPJOINT PTS A' from the user's disk space. The argument IERR contains the CMS return code (error code) that could be checked to ensure the command execution was successful. More detailed documentation on the use of FRTCMX can be found in its listing.

b. The CMS File Identifier

CMS also controls the file identifier. A file created under CMS has an identifier composed of three parts: a file name (maximum of 8 characters), a file type (maximum of 8 characters), and a file mode (2 characters). As an example, in the file identifier:

```
PIPJOINT PTS    A1.
```

PIPJOINT is the file name, PTS is the file type, and A1 (indicating the file is on the user's 'A' disk) is the file mode. CMS does not allow more than one version of a file to exist on a disk space. That is to say, two files on the same

disk space cannot have the same name and file type, which proved to be of some importance in this version's development.⁶

B. THE IBM (CP/CMS) GRAPHICS PACKAGE

As stated earlier, a goal of this version's development was to continue the use of the Tektronix 4000 series terminals for graphics. To this end, the GIFTS graphics package supplied by the University of Arizona was used and required only minor modifications. In addition to these modifications, it was necessary to add four assembly routines to provide for ASCII to EBCDIC conversion and non-interfering I/O operations between the graphics package and the terminal. This section is intended to cover the need for the modifications and the added subroutines. A description of all the graphics package's subroutines can be found in Appendix E, and a listing of the assembly routines can be found in Appendix H.

1. ASCII/EBCDIC Translation

The IBM system uses the EBCDIC character set while the Tektronix terminals use ASCII. To allow communication between the computer and an ASCII terminal, an IBM ASCII interface is provided, within the system, that automatically executes the conversion between ASCII and EBCDIC characters. All I/O operations directed to an ASCII terminal must pass through this interface.

⁶See Section D of this Chapter on Data Base Management.

All output to the terminal from the graphics package is broken up into single characters and placed into an output array named LCHOUT, which is dimensioned 80. These graphics package output characters can be divided into three groups: the graphics characters that are interpreted by the terminal, when in graphics mode, as being coordinates to which the terminal's cursor is to be moved and/or to which a vector is to be drawn; the control characters that place the terminal in alphanumeric mode, graphics mode, erases the screen, rings the bell, etc.; and the text that is to accompany a given plot.

In the GIFTS' graphics package, as with all similar packages, the graphics characters, that is, those characters that provide screen coordinates to the terminal, are computed in ASCII Decimal Equivalent (ADE) integers. These ADE integers, relating directly to ASCII characters, after being calculated, are placed into array LCHOUT for output. If LCHOUT, containing these ADE integers, were output to the terminal, the computer system's ASCII interface would interpret them as being EBCDIC, resulting in improper conversion to ASCII and output to the terminal. Because of this, it is necessary to convert the ADE integers contained in LCHOUT to EBCDIC characters before output to the terminal. This translation, as well as the output to the terminal, is accomplished by the assembly routine WRTADE. WRTADE converts the array of ADE characters to EBCDIC and sends them to the terminal without a carriage return being added to the end of the output

array. For the case where input from the terminal is required by the graphics package, the opposite logic applies. Routine RDADE reads the characters from the terminal, converts them from EBCDIC to ASCII and places them into a desired array for use by the graphics package. More detailed documentation on WRTADE and RDADE can be found in the listing of LIBR5A in Appendix H. Since the graphics characters are calculated in ADE integers and placed into the output array in that form, it is necessary that all other characters placed into the array also be ADE integers. For the control characters, this presents no difficulties since they are already defined in the graphics package in ADE form. The text to accompany the plot, however, is passed to the graphics package as EBCDIC characters in A4 format (up to 4 characters per word). Before the text can be placed into the output array, it must first be broken up into individual characters and converted to ADE integers. This is accomplished by assembly routine TOADE. Additionally, in subroutine CURSOR, following the use of RDADE, it is necessary to translate a character from ADE to EBCDIC for use outside the Graphics Package. The assembly routine TOEBCD is used for this purpose. A listing of both TOADE and TOEBCD can be found in Appendix H.

2. Flushing the Graphics Buffer to the Terminal

When the terminal is in the graphics mode, all characters received are interpreted by the terminal as being either control or graphics characters. If interline characters

are added to the end of the graphics output buffer when 'flushed' to the terminal, they may be interpreted as control characters causing the terminal's mode to be modified, or as graphics characters causing unintended or random vectors to be drawn. As indicated in the previous section, subroutine WRTADE prevents a carriage return from being added to the end of the output buffer, but the IBM system continues to add two characters to the end of the buffer. The first is the control character DC3, and the second the graphics character DEL. Since these characters are added to end of the output buffer, it was necessary to ensure that when the characters are received by the terminal, it would be in alphanumeric mode, thus preventing their interpretation as part of the intended graphics package output. This was accomplished by placing the ADE integer 31 (equivalent to the ASCII control character US), which places the terminal into the alphanumeric mode, into the output array, as the last character.

As a result of placing the terminal into alphanumeric mode at the end of each buffer flush, it was necessary to ensure that the set of graphics characters describing the coordinates for a vector not be split between buffers. If they are split, the desired results may not be obtained. A coordinate description, in ADE form, can consist of up to four characters, so in subroutine PLTCHR, which computes the ADE integers for the desired coordinates, a check is made to ensure that there is room in the output array for at least

four additional characters. If there is not, the array is flushed, which causes the terminal to be left in alphanumeric mode. After this flush, and before PLTCHR continues to calculate the new coordinates, the control character GS (ADE 29), which causes the terminal to switch to graphics mode, is placed into the output array followed by the four control characters that describe the graphic cursor's position prior to the previous flush. It is necessary to provide these coordinates because the first vector drawn after the control character GS is issued to the terminal, is invisible. PLTCHR then continues and calculates the new coordinates and causes them to be placed into the output array.

C. THE IBM (CP/CMS) CHARACTER MANIPULATION PACKAGE

The modifications necessary to the Character Manipulation Package were not extensive in nature and were mainly related to ensuring that the characters used in the many FORTRAN data statements were in EBCDIC form. In addition to these modifications, extensive use was made of five assembly routines provided by the Interactive Graphics Engineering Laboratory of the University of Arizona, for character manipulation. The routines are DECODE, ENCODE, DECOD, BTD, and ENF. A description of all the routines making up the IBM (CP/CMS) Character Manipulation Package can be found in Appendix F and a listing of the assembly routines can be found in Appendix H.

Of importance in this package are the calls to system routines for the date, time of day, and CPU time used. The routine called for the date and time of day is DATIME (called in subroutines DATEP and TIMEP) and for the CPU time used is SETIME (called by subroutine INITIO of the Data Base Management Package) and GETIME (called by subroutine SECOND). If these routines are not available on other systems, appropriate substitutions must be made.

D. THE IBM (CP/CMS) DATA BASE MANAGEMENT PACKAGE

Since the modules communicate with each other through the UDB, it is of the utmost importance that it be correct and managed properly. In addition, it is important that in the management process, that the data base disk space be minimized. This section covers the management of the UDB on the IBM with CP/CMS. A description of all routines used in the IBM (CP/CMS) Data Base Management Package can be found in Appendix G, and a listing of the assembly routines can be found in Appendix H.

1. The IBM (CP/CMS) Data Base

The UDB for the IBM (CP/CMS) version is identical to that of the other versions except for the addition of one file, which will be covered in the SPECIAL SEQUENTIAL DATA BASE FILE section below. The UDB consist of up to 48 random access and 9 sequential files that are managed automatically by GIFTS.

a. The Direct (Random) Access Files

The 48 unformatted direct access UDB files used by the IBM (CP/CMS) version are identical to those used by other versions. It is necessary at this time to introduce some GIFTS terminology in regards to the UDB.

A 'logical record' is the smallest unique collection of data into which the data contained in a file may be divided. As an example, all the data pertaining to one node in the UDB file with the file type PTS make a 'logical record'. The logical record size, in words, is determined by summing the product of the number of 'integers' in the logical record times the number of machine words per integer and the number of 'reals' in the logical record times the number of machine words per real number. Consider again the UDB file with the file type of PTS. It has 10 integers and 17 real numbers per logical record. For single precision, the IBM uses one word for both integer and real numbers, and for double precision, IBM uses one word for integer numbers and two words for real numbers. The word size for a logical record in the PTS file would therefore be 27 for single precision and 44 for double precision.

A 'physical record' is the collection of data that must be read from or written to a file with a single I/O instruction. It may be made up of any integral number of logical records. The number of logical records stored in a physical record of a file is termed the file's 'blocking

factor'. The UDB file type PTS has a blocking factor of 10, and thus a physical record word size of 270 words for single precision and 440 words for double precision. When a physical record is read or written to file type PTS, information on 10 nodes will be involved.

It is the physical record word size that is used in the FORTRAN statement DEFINE FILE to define the random access files. The physical record word size for all the UDB file types can be found in Appendix C. Both single and double precision sizes are included although GIFTS currently is only single precision. It is anticipated that a double precision version will be available in the near future.

The file name assigned to these random access UDB files is the same as the job name provided to GIFTS by the user during module execution. For example, if the user were analyzing a pipe joint, he may provide GIFTS a job name PIPJOINT (the job name, like the file name, is limited to 8 characters), in which case the UDB file type PTS would have the file identifier 'PIPJOINT PTS A'.

b. Sequential Data Base Files

Of the nine sequential files in the UDB, only five will be discussed here, with the remaining four discussed under the heading of SPECIAL SEQUENTIAL DATA BASE FILES below.

These sequential UDB files do not require, as with the direct access files, a specified file size. The file types are CFR, DGT, DYN, HST, and SAV.

The file type CFR is used to store data involved with contours used in module RESULT. File type DGT is used to store digitizer information, while DYN stores information relating to modal analysis, with HST containing information for a histogram plot in transient analysis. File SAV is used to save the stiffness matrix.

These files are automatically created by the GIFTS modules and are given a file name equal to the GIFTS job name.

c. Special Sequential Data Base Files

These files are placed in this category because they do not follow the naming convention of other UDB files and are not necessarily created by the GIFTS modules.

The first of these special UDB files has the file type SRC. An SRC file can be created with any file name, by the user, via the text editor in the CMS environment and will contain a list of GIFTS commands and their associated data. The commands can be executed from an interactive module via the command OLB (On Line Batch). When the OLB command is executed GIFTS prompts the user for the file name of the SRC file. After the file name is entered, the commands contained in the file are read and executed by GIFTS. For more information on the SRC file's use, consult the GIFTS User's Reference Manual [Ref. 1].

The next special file has the identifier GIFTS5 INF and resides on the GIFTS system's disk space. This file

contains a listing of all GIFTS' commands with instructions for their use. The file is accessed from interactive modules via the command HELP. When the command is issued, GIFTS prompts the user for the command for which he desires information. After this is entered, GIFTS opens the file, locates the desired information, and displays it at the terminal for the user.

The last two special files are GIFTS5 EST and GIFTS5 ESX which generally reside on the user's disk space. GIFTS5 EST contains information which is used by solution module OPTIM to make solution time estimates for the solution modules STIFF, DECOM, DEFL, and STRESS. These four modules perform updates to the file during their execution. GIFTS logic is such that during this updating procedure, it performs both read and write operations to GIFTS5 EST. In other computer systems on which GIFTS is operated, this is accomplished by opening one GIFTS5 EST file for input and another for output, on the same disk space. Since the IBM system does not allow two files to exist on the same disk space with the same identifier, this presented a problem. This problem was solved by opening file GIFTS5 EST for input, but having output directed to GIFTS5 ESX. This is accomplished when GIFTS calls for the opening of GIFTS5 EST for output. The subroutine that opens sequential files for output, changes the file type from EST to ESX. When these modules complete their I/O operations on these files, and the files are closed, GIFTS5 EST, now being

superseded by GIFTS5 ESX, is deleted from the user's disk space, so that the user will only have one of these files present on his disk space. When the next GIFTS module is executed, GIFTS5 ESX is renamed GIFTS5 EST, making it ready for input. If the user executes one of the solution modules without GIFTS5 EST or GIFTS5 ESX being present on his disk space, GIFTS will use the GIFTS5 EST file on the GIFTS system's disk space for input and will create GIFTS5 ESX on the user's disk space during output.

2. Defining the Direct Access Files and GIFTS Module IBMUDB

Before unformatted direct access I/O operations can occur, the direct access file must first be defined. This is accomplished with the FORTRAN statement:

```
DEFINE FILE ISL (INR,ISR,U,IV),
```

where: ISL is the logical unit number assigned to the file,
INR is the maximum number of 'physical records' that can be contained in the file,
ISR is the 'physical record' size in words,
U indicates that the records are to be written and read without format control, and
IV is the record number associated variable (not used in GIFTS but must be included in the statement).

In all the versions of GIFTS that have been available, the machine's FORTRAN allowed ISL, INR, and ISR to be integer variables. The logics and structure of the GIFTS data base

management was based on this. This fact allowed the use of one DEFINE FILE statement, located in subroutine DEFIN of LIBR5, in these versions. The logical unit number (ISL), the 'physical record' size in words (ISR), and the number of 'physical records' required (INR), were simply passed as arguments to this subroutine. When it became necessary to increase the size of the file, that is increase the number of records allowed in the file (INR), the file was closed via a call to a system function or through the use of a FORTRAN statement, and then the file would be redefined with the same value for ISR and the increased value for INR. The logical unit number (ISL) for the file could change as the file was opened and closed.

This logic presented a problem for the IBM (CP/CMS) version, since both the IBM FORTRAN G and FORTRAN H-extended, available at the Naval Postgraduate School, do not allow ISL, INR, and ISR to be integer variables. They must be integer constants. Additionally, since there was no way to close a file during execution⁷, once the DEFINE FILE statement was made in an executing program, it could not be changed in that program to allow for file growth.

It became clear that a DEFINE FILE statement would be required for each direct access file. For this, logical unit

⁷The CMS command FINIS only closes a file in the CMS environment, not in the FORTRAN execution environment (IBCOM).

numbers 50 through 98 were used and the 'physical record' size, in words, was calculated as described in the preceding section on direct access files. This left only the number of 'physical records' (INR) to be entered, and as the only undetermined quantity, with its value dependent on the desire to conserve disk space and to ensure that the user would not be unknowingly constrained by a data base too small for a desired analysis. Both of these seemingly opposite requirements were met through the introduction of the new GIFTS module IBMUDB.

Before continuing with more information on program IBMUDB, it is necessary to explain some facts about direct access file creation on the IBM system and about some additional GIFTS' logic and structure.

Consider the FORTRAN statement:

```
DEFINE FILE 10 (100,27,U,IV).
```

It defines a direct access, unformatted file on logical unit 10. Each 'physical record' will be 27 words long and there may be up to 100 'physical records' written to the file. The statement itself does not cause the creation of this file. It is only after the file is written into that this occurs. If the file did not exist prior to program execution, and only 5 records are written to the file, the system will write those 5 records, plus fill the remaining 95 records with zeros. The result is a great deal of wasted disk space. However, if the file existed on the disk space

prior to the program execution containing this DEFINE FILE statement, there could be a different result. As an example, assume that the file in question was created by a previous program execution containing the statement:

```
DEFINE FILE 10 (1,27,U,IV).
```

This is similar to the previous DEFINE FILE statement except that only one record can be written (or read), as indicated by a value of 1 for INR. When the program containing the DEFINE FILE statement allowing 100 records is executed and the file written into, only those records written will be added to the file. As an example, if during the second program's execution, the same five records are written to the file, only those five records will be written to the disk, not the 100 as before, thus saving disk space. It can be seen that this method, if properly used, could result in the conservation of disk space while allowing for file growth.

It is now necessary to discuss some important facts about GIFTS and its data base management logic. Whenever one of the GIFTS model generation modules⁸ is first executed for an analysis, it checks for the presence of the data base files for the given job name, and if they are present, GIFTS calls for the deletion of the old data base and the creation of a new one. If the data base for an analysis was created

⁸Modules BULKM, BULKS, EDITM, and EDITS.

prior to the execution of one of these modules, for the purpose of conserving disk space and allowing for file growth, as described above, its intended purpose would be defeated if it were deleted.

The new GIFTS module IBMUDB was developed to create a 'dummy' data base for GIFTS and is executed prior to performing an analysis on the IBM. In IBMUDB, each direct access data base file is defined with its respective 'physical record' size, in words, and with a maximum of one record. Then, in the first word of that record, the integer -999 is written. Thus, a data base file with -999 in the first word of the first record is considered a 'dummy' data base file, and through the modification of the Data Base Management Package of LIBR5, GIFTS considers a 'dummy' file not to be present. A listing of module IBMUDB can be found in Appendix D.

The DEFINE FILE statements executed in all modules, except IBMUDB, are located in subroutine INITIO of the Data Base Management Package of LIBR5. In these statements, each direct access data base file is defined with its respective logical unit number and 'physical record' size in words, and with a maximum number of records of 1,000,000, which essentially provides the user unlimited growth for the data base. The actual data base size will only be limited by the user's disk space available.

3. Opening of the Data Base Files

On the IBM system, a file is opened automatically when an I/O operation⁹ is performed, using the file's logical unit number. If the logical unit number has not been associated with a file identifier by the user, CMS automatically assigns an identifier based on the logical unit number used. As an example, if the sequential I/O statement:

WRITE (29,100),

were executed without the association, CMS would assign the identifier:

FILE FT29F001 A,

which tells nothing about the file contents. In keeping with the original structure of GIFTS, the opening of a file was considered to be the association of the proper file identifier with its logical unit number.

In other versions of GIFTS, a maximum of thirteen files (ten direct access and three sequential) were allowed to be open at any given time. Generally, logical unit numbers 1 through 4 and 7 through 12 were used for the direct access files while 13 through 15 were used for the sequential files. The logical unit numbers were assigned by subroutine SLTASN

⁹For direct access files the DEFINE FILE statement must precede the I/O operation. Additionally, if the operation is for input, the file must already exist.

of the Data Base Management Package. When necessary, the logical unit number would be released by the closing of the file and then the freeing of the number by subroutine FRESLT in LIBR1. This logic works provided that during the execution of a program, a logical unit number could be used for more than one direct access file identifier, which is not the case for the IBM system with FORTRAN G or FORTRAN H-extended. On the IBM, once a logical unit number is used for a given direct access file, it cannot be associated with any other file. Because of this, in the IBM (CP/CMS) version, each UDB file is assigned its own logical unit number. Subroutine FRESLT, which was in LIBR1 and thus considered a machine independent routine, was moved to LIBR5 and made a 'dummy' routine, that is, it was changed to contain no executable FORTRAN statements. Subroutine SLTASN was also made a 'dummy' routine. These two routines were not deleted from GIFTS because they are called by routines contained in the machine independent portions of GIFTS and it was desired not to alter these. To compensate for the void left by making SLTASN a 'dummy' routine, subroutine MANAGE was added to the Data Base Management Package. This routine relates a data base file type with its respective logical unit number. It will return the logical unit number if provided with the file type or the file type if provided with the logical unit number. MANAGE is called only by routines contained in the Data Base Management Package. The logical unit numbers for each of the data base files can be found in Appendix C.

All the data base files are opened, that is the data base file identifier is associated with its respective logical unit number, by invoking the CMS command FILEDEF during program execution,¹⁰ when called for by GIFTS. For direct access files, this is accomplished in subroutine DEFIN. Here the FILEDEF command is issued and has the form:

```
FILEDEF ISLDEF DISK XJOB EXT8 A (XTENT 1000000 DSORG DA)
```

where: ISLDEF is a double word real variable containing the logical unit number,

DISK indicates the file is to be disk resident,

XJOB is a double word real variable containing the file name,

EXT8 is a double word real variable containing the file type,

A is the file mode,

XTENT 1000000 is the maximum number of records in the extent for the file, and,

DSORG DA indicates that the data set organization is direct access.

For the sequential files, the statement:

```
FILEDEF ISLDEF DISK XJOB EXT8 A
```

is generally used, with ISLDEF, DISK, XJOB, EXT8, and A having the same meaning as for the direct access FILEDEF.

¹⁰This is accomplished via a call to assembly routine FRTCMX discussed in Section A of this Chapter.

Three subroutines, OPENIF, OPENOF, and OPENLP are used to open the sequential files. Subroutine OPENOF opens a file for output and uses a FILEDEF statement identical to the one above. Subroutine OPENIF is used to open a sequential file for input. In this routine, if the file to be opened is either GIFTS5 EST or GIFTS5 INF,¹¹ a check is made to determine if the file exists on the user's disk, if it does not, the FILEDEF for the file is made with the file mode set to C, where C indicates that the file is to be read from the GIFTS system's disk space to which the user is linked. Subroutine OPENLP opens a sequential file, for output, that may be spooled to the line printer by GIFTS. The FILEDEF used here is:

FILEDEF 20 DISK PRTNAM PRNTFILE A,

where: 20 is the logical unit number,
DISK indicates that the file is to be disk
 resident,
PRTNAM is a double word real variable containing
 the file name, provided by the user,
PRNTFILE is the file type, and,
A is the file mode.

4. Closing Files

Since the IBM FORTRAN G and FORTRAN H-extended do not provide FORTRAN statements that allow a file to be closed

¹¹See section entitled Special Sequential Access Files in this Chapter.

during execution, the IBM automatically closes all files opened during execution when the execution is terminated. This, coupled with the fact that GIFTS has not been limited as to the number of files it may have opened during execution (see the previous section), it may at first appear that there is no need to be concerned about closing a file, which is in fact the case in the FORTRAN execution environment, IBCOM. However, in the CMS environment, in order to rename a file, it must be inactive (closed). There is a CMS command, FINIS, that does allow for the closing of a file in the CMS environment. This command is invoked from GIFTS during execution via a call to FRTCMX (see Section A of this Chapter) in subroutine CLOSEF of the Data Base Management Package of LIBR5. This command has been included in CLOSEF for the express purpose of allowing a file to be renamed, which will be discussed later in this Chapter. The FINIS command issued in CLOSEF has the form:

FINIS XJOB EXT8 A,

where XJOB and EXT8 are double word real variables containing the file name and file type, respectively, and A is the file mode.

There is an additional subroutine called by GIFTS to close a file. It is subroutine CLOSLP, which is intended to close the line printer file and spool it to the line printer as directed by the user. For this version, the system was allowed to close the file at the termination of module

execution. When the subroutine is called, it will, if a printer file has been created, prompt the user if he desires the file to be spooled to the line printer or not. In either case, the file will remain on the user's disk space following execution termination. The CMS command used here is:

```
PRINT PRTNAM PRNTFILE A,
```

where: PRTNAM is a double word real variable containing the file name,
PRNTFILE is the file type, and
A is the file mode.

5. Deleting Files

Based on the discussion in the section entitled DEFINING THE DIRECT ACCESS FILES AND GIFTS MODULE IBMUDB, earlier in this Chapter, it is obvious that deletion of a direct access data base file would defeat the logic behind the use of module IBMUDB. When GIFTS calls subroutine DELETE for the deletion of that file, rather than being deleted, it is made into a 'dummy' file by writing -999 into the first word of the first record of the file. A 'dummy' file will be interpreted by the Data Base Management Package as not being present. Subroutine DELETE does delete sequential files via the CMS command ERASE. This command in DELETE has the form:

```
ERASE XJOB EXT8 A
```

where XJOB, EXT8 and A are the same as those for the FINIS command in the previous section.

6. Renaming Files

During the execution of some of the GIFTS modules, temporary files are created and later renamed. Specifically, the data base file types PTX, LDX, and SLX are created and later renamed PTS, LDS, and SLI, respectively. To understand better the need to do this, and the logic of GIFTS in this matter, consider the file type PTS. This file contains all the information pertaining to the nodes of the model and is first created during model generation. The nodes are entered in the file in numerical order, based on user assigned numbers. Solution module OPTIM is executed to optimize the node numbering in order to decrease the problem 'band width'. In this process, the nodal information is read in from file type PTS, and the nodes are renumbered and output in numerical order, based on the optimization renumbering, to file type PTX. Once this has been accomplished, OPTIM calls for file type PTS, of the current job, to be deleted and for the renaming of file type PTX, of the current job, to be renamed to file type PTS. The renaming is accomplished in subroutine RENAME of the Data Base Management Package. Since direct access files are not actually deleted (see the preceding section on the deleting of files), but rather made into 'dummy' files, both PTS and PTX will actually always be present on the disk. In order to maintain both files on the disk, what is actually desired is an exchange of names between PTS and PTX. This exchange of names is accomplished via the CMS command RENAME with the form:


```

RENAME XJOB PTS A XJOB XXX A
RENAME XJOB PTX A XJOB PTS A
RENAME XJOB XXX A XJOB PTX A

```

Here, the file with the identifier XJOB PTS A is renamed XJOB XXX A, then file XJOB PTX A is renamed XJOB PTS A and finally, file XJOB XXX A is renamed XJOB PTX A. This 'around about' method is necessary due to the fact that no two files can exist on a CMS managed disk with the same identifier.

In subroutine RENAME, the actual RENAME command has the form:

```

RENAME XJOB EXT81 A XJOB XXXXX A
RENAME XJOB EXT82 A XJOB EXT81 A
RENAME XJOB XXXXX A XJOB EXT82 A

```

where: XJOB is a double word real variable containing the file name,

EXT81
and are double word real variables, each con-
EXT82 taining one of the file types to be renamed,

XXXXX is the intermediate file type 'XXX' for the name exchange process, and

A is the file mode.

There is an additional complication to the renaming process resulting from the inability to close a file in the FORTRAN execution environment (IBCOM) of the IBM system. Because of this, once an I/O operation has been performed to a direct access file with its respective logical unit number, that logical unit number cannot be changed. Again consider the file types PTS and PTX. Their logical unit numbers in

this version are 86 and 87 respectively. If, during the execution of OPTIM, after the files have been renamed, an I/O operation is directed at file type PTS (the old PTX file), logical unit number 87 (the logical unit number old file PTX) must be used. Since subroutine MANAGE¹² assigns logical unit numbers based on file types, it was necessary that MANAGE be aware when renaming occurred, so that the renamed files would maintain their original logical unit numbers. This was accomplished by setting a switch, passed to MANAGE by a labeled common block, in RENAME for the files that are renamed.

It was also necessary, in this IBM (CP/CMS) version to rename, if it exists, file GIFTS5 ESX A to GIFTS5 EST A¹³ at the start of module execution. This is accomplished by invoking the CMS RENAME command in subroutine INITIO of the Data Base Management Package.

7. Checking for a File's Presence

GIFTS utilizes logical function PRESNT of the Data Base Management Package to check for the presence of a UDB file for use by GIFTS. PRESNT first determines if the file is present on the user's disk space via the CMS command:

STATE XJOB EXT8 A.

¹²See the Section on the Opening of Files in this Chapter.

¹³See the section on Special Sequential Data Base Files in this Chapter.

If the file is sequential and is determined to exist on the user's disk space, PRESNT is set equal to TRUE, if not on the disk space, then it would be set to FALSE. For a direct access file, if the file exists on the disk space, the first word of the first record is read and if it does not equal -999, PRESNT is set equal to TRUE, and if it does equal -999, it is a 'dummy' file and PRESNT is set equal to FALSE. If, on the other hand, the file is direct access and is not present on the user's disk space, PRESNT stops module execution and issues the message:

UDB FOR JOB XXXXXXXXX NOT CREATED VIA IBMUDB.

where XXXXXXXXX will be the job name provided GIFTS by the user.

E. THE ADDED ASSEMBLY ROUTINES (LIBR5A)

For convenience, all the assembly routines for this version of GIFTS have been placed, as entry points, into LIBR5A. A listing of LIBR5A can be found in Appendix H.

F. IBM (CP/CMS) INSTALLATION INSTRUCTION

The installation of GIFTS on the IBM involves the transferring of the source code from tape to the CMS disk space, the compiling of the code, its loading (linking) and execution. This section covers the steps necessary to install GIFTS on the IBM system at the Naval Postgraduate School. Installation on other systems should vary only slightly.

The GIFTS source code is received from the University of Arizona on an unlabeled nine track tape. Accompanying the tape is a tape directory indicating the names and sizes of the files and the order in which they were written onto the tape.

At the Naval Postgraduate School, all but one of the tape drives are attached to MVS (Multiple Virtual System - the batch operating system). Because of this, it has proven expedient to first transfer the files to an MVS disk and then to the CMS disk space. Transfer from the tape to the MVS disk is accomplished via the IBM OS Utility IEBGENER. A listing of the file TAPE2MVS JCL (which resides on the GIFTS disk space), containing the necessary job control statements¹⁴ for the transfer via IEBGENER can be found in Appendix I. These job control statements will cause the first 37 files on the tape to be transferred to disk MVS004 with the respective file identifiers S3161.FILE1 through S3161.FILE37. To transfer the files to the CMS disk space, it is first necessary to link to and access MVS004. This is accomplished by issuing the commands:

```
CP LINK MVS 36B 36B RR
ACCESS 36B E.
```

Once this is done, the files can be transferred to CMS, renamed in accordance with the tape directory, and packed, by

¹⁴These job control statements are for an 800 bpi ASCII tape.

the use of the CMS commands listed in Appendix I. At the Naval Postgraduate School, these commands are contained in file MVS2CMS EXEC (on the GIFTS disk space) and can be executed by issuing the command MVS2CMS. For future updates, if the number, name, or order of the files to be transferred differ from those now listed in TAPE2MVS JCL and MVS2CMS EXEC, appropriate changes should be made.

All the FORTRAN program modules and libraries, except BULKM and BULKS, can be compiled with the IBM FORTRAN G compiler. It will be necessary, due to variations in the FORTRAN used, to compile BULKM and BULKS with the IBM FORTRAN H-extended compiler.

Following compiling, the program modules and libraries are ready for loading and execution. This, as an example, can be accomplished for program module BULKM, via the CMS commands:

```
LOAD BULKM LIBR1 LIBR2 LIBR3 LIBR4 LIBR5 LIBR5A
START.
```

At the Naval Postgraduate School, this is accomplished by the use of an exec file (see Appendix H).

G. IBM (CP/CMS) USER INSTRUCTIONS

The user's instructions for the IBM (CP/CMS) version vary only slightly from those contained in the GIFTS User's Reference Manual [Ref. 1]. Appendix J contains all necessary additional information to use this version at the Naval Postgraduate School.

IV. SUMMARY AND RECOMMENDATIONS

In summary, the IBM (CP/CMS) version of GIFTS developed is a self-contained, portable system that will allow easy installation on other IBM computers with the CMS operating system.

It is recommended that additional work be done, at the Naval Postgraduate School, on the Graphics Package to enable the use of the Tektronix 618 graphics terminals being installed on the IBM system.

APPENDIX A:
DESCRIPTION OF GIFTS MODULES

IBM DATA BASE INITIALIZATION

IBMUDB A program to create the dummy¹⁵ data base for the
 IBM (CP/CMS) version of GIFTS.

MODEL GENERATION AND EDITING

BULKM An automated three dimensional plate and shell
 model generator. It is suitable for large contin-
 uous structures that can be easily modeled by
 repetitious generation of points and elements.

BULKS A three dimensional solid model generator. One
 may ask for the display of the edges, and may add,
 and display selected point and element slices.

EDITM Designed to update and correct BULKM models,
 although it can be used to generate simple models
 and ones too complex for BULKM.

EDITS A three dimensional solid mesh editor. It may be
 used to update or correct BULKS models, or to gen-
 erate simple models and those too complex for
 BULKS.

¹⁵See Chapter III, Section D, Subsection 2.

LOAD AND BOUNDARY CONDITION GENERATION, DISPLAY AND EDITING

- BULKF Suppress those freedoms which the generated model cannot support, thereby relieving the user of the necessity of suppressing all superfluous freedoms by hand.
- BULKLB The bulk load and boundary condition generator designed to apply loads to models generated with BULKM. It may be used to apply distributed line and surface loads and masses, prescribed displacements along lines and surfaces, and inertial loads. Temperatures may also be applied to lines and surfaces.
- LOADS The load and boundary condition generator for solid models. Loads may be distributed on lines or surfaces. Loads and boundary conditions may be displayed on point slices.
- EDITLB A display and edit routine intended to provide local modification capability to loads and boundary conditions applied by BULKLB. It may also be used to generate simple loading on models, or loading on models not generated with BULKM. Temperatures may also be applied to elements. After DEFL has been run, the thermal and combined loads may be examined.

GENERAL PURPOSE COMPUTATIONAL AND RESULT DISPLAY MODULES

OPTIM Band width optimization program. OPTIM may be called several times in a row, until the best node numbering scheme has been achieved.

STIFF Computation of element stiffness matrices and their assembly into the master stiffness matrix.

SAVEK This program saves the newly computed stiffness matrix.

PRINT Program used to print out parts of the stiffness matrix and directory for specified GIFTS models.

DECOM Introduces kinematic boundary conditions, and decomposes the stiffness matrix using the Cholesky method.

DEFL Computation of the deflections from the current loading conditions and the decomposed stiffness matrix. If temperatures are present, thermal forces will be calculated and added to the current applied loads before solution.

STRESS Computation of element stresses based on current deflections.

RESULT Result display. The program displays deflections and stresses. It has many options that may be used, at the discretion of the user, to transform the results for optimum comprehension.

RESIDU This program computes the residual forces on a structure solved by the GIFTS solution programs.

THE GIFTS NATURAL VIBRATION PACKAGE

AUTOL Used to generate starting loads for the subspace iteration to compute natural modes of vibration.

SUBS Performs a single subspace iteration to determine the model's natural modes. It must be repeated as many times as necessary to obtain convergence to the desired extent.

THE GIFTS TRANSIENT RESPONSE PACKAGE

TRAN1 Used to specify the time step to be used in the integration process and is to be run on a transient response model immediately after stiffness assembly.

TRAN2 Computes the displacement matrix for time T and is run after TRAN1 and DECOM.

TRANS Maintains and plots histograms of the displacements of up to four different freedoms.

THE GIFTS CONSTRAINED SUBSTRUCTURING PACKAGE

DEFCS Program to define a constrained substructure's (COSUB) boundaries and external nodes.

REDCS This program generates the reduced stiffness and loads matrices necessary for assembly of the 'COSUB' in the main analysis, as well as the transformation matrices necessary for local analysis.

LOCAL This program performs local analysis of selected
 'COSUB' models from a major analysis.

THE GIFTS LIBRARIES

LIBR1

LIBR2

LIBR3

LIBR4 Commonly used machine independent subroutines and
 functions.

LIBR5 Machine dependent FORTRAN subroutines and functions
 used for graphics, character manipulation and data
 base management.

LIBRD5A Collection of CP/CMS assembler routines used by
 LIBR5 of the IBM (CP/CMS) version.

APPENDIX B:
SUMMARY OF CMS COMMANDS USED

COMMANDS USED IN PROGRAM PREPARATION

EDIT	Used to invoke the VM system product editor to create or modify a CMS disk file.
COPYFILE	Used to copy CMS disk files according to given specifications. Used in GIFTS implementation to 'pack' FORTRAN source code to conserve disk space.
FORTG1	Used to compile FORTRAN source code using the G1 compiler.
FORTHX	Used to compile FORTRAN source code using the H-extended compiler.
TXTLIB	Used to generate and modify TEXT libraries.
LOAD	Used to bring TEXT files into storage for execution.
START	Used to begin execution of programs previously loaded into storage.

COMMANDS USED IN GIFTS FOR FILE MANIPULATION

ERASE	Used to delete CMS disk files.
FILEDEF	Used to relate logical unit numbers, devices and files.
FINIS	Used to close files in the CMS environment.

PRINT	Used to spool a specified CMS file to the virtual printer.
RENAME	Used to change the names of CMS files.
STATE	Used to verify the existence of CMS disk files.

APPENDIX C:
THE UNIFIED DATA BASE FOR THE IBM

SEQUENTIAL DATA BASE FILES

<u>FILE TYPE</u>	<u>LOGICAL UNIT NUMBER</u>
CFR	27
DGT	23
DYN	28
HST	25
SAV	24

SPECIAL SEQUENTIAL DATA BASE FILES

<u>FILE TYPE OR IDENTIFIER</u>	<u>LOGICAL UNIT NUMBER</u>
SRC	26
GIFTS5 INF	21
GIFTS5 EST	22
GIFTS5 ESX	29

THE RANDOM ACCESS FILES

<u>FILE TYPE</u>	<u>NUMBER OF INTEGERS</u>	<u>NUMBER OF REALS</u>	<u>BLOCKING FACTOR</u>	<u>IBM SP WORDS</u>	<u>IBM DP WORDS</u>	<u>LOGICAL UNIT NUMBER</u>
CDN	4	180	1	184	364	51
CEE	4	324	1	328	652	56
CLD	4	180	1	184	364	52
DNH	0	6	10	60	120	64
DNI	4	180	1	184	364	53
DNS	0	8	10	80	160	65
DNT	2	72	1	74	146	75
ELD	6	32	5	190	350	77
ELS	0	12	10	120	240	78
ELT	25	10	10	350	450	79
FIL	5	1	1	6	7	98
GRD	43	10	5	265	315	80
INT	5	0	10	50	50	81
KEE	4	324	1	328	652	57
LD2	0	8	10	80	160	66
LDC	0	8	10	80	160	67
LDI	4	180	1	184	364	54
LDS	0	8	10	80	160	68
LDT	2	72	1	74	146	76
LDX	0	8	10	80	160	69
LIN	31	9	10	400	490	82
MAT	5	11	5	80	135	72

<u>FILE TYPE</u>	<u>NUMBER OF INTEGERS</u>	<u>NUMBER OF REALS</u>	<u>BLOCKING FACTOR</u>	<u>IBM SP WORDS</u>	<u>IBM DP WORDS</u>	<u>LOGICAL UNIT NUMBER</u>
MEE	4	324	1	328	652	58
OP0	2	0	40	80	80	73
OP1	4	0	40	160	160	83
OP2	2	0	40	80	80	74
OP3	1	0	40	40	40	84
PAR	25	0	1	25	25	97
PLD	6	8	10	140	220	85
PTS	10	17	10	270	440	86
PTX	10	17	10	270	440	87
RES	0	8	10	80	160	70
SDY	42	0	5	210	210	88
SD2	21	0	5	105	105	89
SLD	129	9	1	138	147	90
SLI	18	0	10	180	180	91
SLX	18	0	10	180	180	92
STC	4	324	1	328	652	59
STF	4	324	1	328	652	60
STR	0	8	10	80	160	71
SUR	4	26	5	150	280	93
TDE	4	324	1	328	652	61
TEM	4	324	1	328	652	62
THS	6	15	5	105	180	94
TIE	4	324	1	328	652	63

<u>FILE TYPE</u>	<u>NUMBER OF INTEGERS</u>	<u>NUMBER OF REALS</u>	<u>BLOCKING FACTOR</u>	<u>IBM SP WORDS</u>	<u>IBM DP WORDS</u>	<u>LOGICAL UNIT NUMBER</u>
TLD	0	84	5	420	840	95
TMP	0	28	5	140	280	96
UGC	4	180	1	184	364	55

APPENDIX D: LISTING OF NEW GIFTS PROGRAM MODULE IBMUDB

```
C ::: THIS PROGRAM CREATES THE DUMMY DATA BASE FOR THE GIFTS IBM(CP/CMS)
C C NPS 4 JAN 82 LT R. HUNDLEY, USN
C C
REAL*8 ISLDEF,EXT(48)
DIMENSION XJOB(2)
DATA EXT/,CDN
1 2 3 4 5 6 7 8 9 10 11 12
LDI
MEE
TEM
LD2
RES
OP2
ELS
LIN
PTS
SLD
THS
FIL
DNI
KEE
TDES
DNS
LDX
OPO
ELDT
INT
PLD
SD2
SUR
PAR
CLD
CEE
UGC
STF
DNH
LDS
MAT
LDT
GRD
OP3
SDY
SLX
TMP
WRITE (6,100)
READ (5,101) XJOB
WRITE (6,103) XJCB
READ (5,105) ANS
IF (ANS.EQ. YES) GO TO 12
GO TO 11
CALL FRTCMS ('ERASE ',XJOB,'*'
ISL=I
CALL CONVRT (ISL,ISLDEF)
J=I-9
CALL FRTCMS ('FILEDEF ',ISLDEF,'DISK ',XJOB,EXT(J),','
,XTEXT ',1',DSORG ',DA
10 CONTINUE
FOR CDN,CLD,DNI,LDI,UGC
DEFINE FILE 10 ((1,184,U,IV)
DEFINE FILE 11 ((1,184,U,IV)
DEFINE FILE 12 ((1,184,U,IV)
DEFINE FILE 13 ((1,184,U,IV)
DEFINE FILE 14 ((1,184,U,IV)
FOR CEE,KEE,MEE,SIC,STF,TDE,TEM,TIE
DEFINE FILE 15 ((1,328,U,IV)
DEFINE FILE 16 ((1,328,U,IV)
DEFINE FILE 17 ((1,328,U,IV)
DEFINE FILE 18 ((1,328,U,IV)
DEFINE FILE 19 ((1,328,U,IV)
DEFINE FILE 20 ((1,328,U,IV)
DEFINE FILE 21 ((1,328,U,IV)
```


C	::	DEFINE FILE	22	(1,328,U,IV)	IBM00480
		FOR DNH			IBM00490
C	::	DEFINE FILE	23	(1,60,U,IV)	IBM00500
		FOR DNS,LD2,LD3			IBM00510
		DEFINE FILE	24	(1,80,U,IV)	IBM00520
		DEFINE FILE	25	(1,80,U,IV)	IBM00530
		DEFINE FILE	26	(1,80,U,IV)	IBM00540
		DEFINE FILE	27	(1,80,U,IV)	IBM00550
		DEFINE FILE	28	(1,80,U,IV)	IBM00560
		DEFINE FILE	29	(1,80,U,IV)	IBM00570
		DEFINE FILE	30	(1,80,U,IV)	IBM00580
		DEFINE FILE	31	(1,80,U,IV)	IBM00590
		DEFINE FILE	32	(1,80,U,IV)	IBM00600
		DEFINE FILE	33	(1,80,U,IV)	IBM00610
C	::	FOR DNT,LD1			IBM00620
		DEFINE FILE	34	(1,74,U,IV)	IBM00630
		DEFINE FILE	35	(1,74,U,IV)	IBM00640
C	::	FOR ELD			IBM00650
		DEFINE FILE	36	(1,190,U,IV)	IBM00660
C	::	FOR ELS			IBM00670
		DEFINE FILE	37	(1,120,U,IV)	IBM00680
C	::	FOR ELT			IBM00690
		DEFINE FILE	38	(1,350,U,IV)	IBM00700
C	::	FOR GRD			IBM00710
		DEFINE FILE	39	(1,265,U,IV)	IBM00720
C	::	FOR INT			IBM00730
		DEFINE FILE	40	(1,50,U,IV)	IBM00740
C	::	FOR LIN			IBM00750
		DEFINE FILE	41	(1,400,U,IV)	IBM00760
C	::	FOR OP1			IBM00770
		DEFINE FILE	42	(1,160,U,IV)	IBM00780
C	::	FOR OP3			IBM00790
		DEFINE FILE	43	(1,40,U,IV)	IBM00800
C	::	FOR PLD			IBM00810
		DEFINE FILE	44	(1,140,U,IV)	IBM00820
C	::	FOR PTS,PTX			IBM00830
		DEFINE FILE	45	(1,270,U,IV)	IBM00840
		DEFINE FILE	46	(1,270,U,IV)	IBM00850
C	::	FOR SDY			IBM00860
		DEFINE FILE	47	(1,210,U,IV)	IBM00870
C	::	FOR SD2			IBM00880
		DEFINE FILE	48	(1,105,U,IV)	IBM00890
C	::	FOR SLD			IBM00900
		DEFINE FILE	49	(1,138,U,IV)	IBM00910
C	::	FOR SLI,SLX			IBM00920
		DEFINE FILE	50	(1,180,U,IV)	IBM00930
		DEFINE FILE	51	(1,180,U,IV)	IBM00940
C	::	FOR SUR			IBM00950


```

C ::: DEFINE FILE 52 (1,150,U,IV)
FOR THS
C ::: DEFINE FILE 53 (1,105,U,IV)
FOR TLD
C ::: DEFINE FILE 54 (1,420,U,IV)
FOR TMP
C ::: DEFINE FILE 55 (1,140,U,IV)
FOR PAR
C ::: DEFINE FILE 56 (1,25,U,IV)
FOR FIL
C ::: DEFINE FILE 57 (1,6,U,IV)
IDUMY=-999
DO 20 I=10,57
20 WRITE (1,1) IDUMY
WRITE (6,102) XJOB
100 FORMAT (1X,ENTER JOB NAME)
101 FORMAT (2A4)
102 FORMAT (1X,GIFTS UNIFIED DATA BASE FOR JOB,1X,2A4,1X,CREATED..)
103 FORMAT (1X,THIS WILL DESTROY ALL FILES NAMED,1X,2A4,1X,
/1X,DO YOU WISH TO CONTINUE (Y/N)?)
1 STOP
11 WRITE (6,104) XJCB
104 FORMAT (1X,GIFTS UDB FOR JOB,1X,2A4,NOT CREATED..)
105 FORMAT (A1)
STOP
END

```

```

IBM000960
IBM000970
IBM000980
IBM000990
IBM010000
IBM010100
IBM010200
IBM010300
IBM010400
IBM010500
IBM010600
IBM010700
IBM010800
IBM010900
IBM011000
IBM011100
IBM011200
IBM011300
IBM011400
IBM011500
IBM011600
IBM011700
IBM011800
IBM011900
IBM012000
IBM012100

```


APPENDIX E:

THE IBM (CP/CMS) GRAPHICS PACKAGE SUBROUTINES (* indicates a modified or added subroutine)

ANMODE Places the Tektronix 4000 series graphics terminal into the alphanumeric mode.

CURSOR * Turns on the graphics cursor, and reads back the x and y coordinates of the cursor position when a character is struck on the key board. It also returns the value of the character struck. This subroutine calls assembly routine RDADE for the input operation and TOEBCD for conversion of the character to EBCDIC.

DINIT This subroutine clears the terminal screen and positions the graphics cursor at (0,0) in the viewing area.

DRAW Draws a visible line from the current beam position to the absolute coordinates provided.

FLUSH * This subroutine dumps the graphics output array to the terminal and insures that the last character in the array is US, which places the terminal into alphanumeric mode preventing the interpretation of interline characters, sent by the IBM system, as graphics characters. Assembly routine WRTADE is used to dump the buffer.

INTERP	This function linearly interpolates point coordinates for graphics output.
LINA	This subroutine will draw an invisible or visible line, as directed, from the current graphics cursor position to the absolute coordinates provided.
LINE	This subroutine will draw an invisible or visible line, as directed, from the current graphics cursor position to the absolute coordinates provided.
MOVE	Subroutine that invisibly moves the terminal's graphics cursor to the absolute screen coordinates provided.
OFFSET	This subroutine switches the Graphics Package between the 800 x 800 main viewing area and the 800 x 223 offset viewing area.
PLTCHR *	This subroutine computes, and outputs, the graphic control characters necessary to draw a line from the current beam position to the absolute screen coordinates provided. It will send the minimum number of characters needed, from one to four. If the output array does not have room for at least four characters, it is flushed to the terminal. Following the flush, the terminal is placed back into graphics mode and the cursor moved back to its position prior to the flush.

PUTCHR This subroutine adds characters to the output array. If number of characters in the array reaches NCHMAX, the array is flushed.

SETPT This subroutine invisibly positions the graphics beam to the absolute screen coordinates provided.

TABLET * This subroutine reads one point from the digitizing tablet. It is currently inactive in this version.

TBELL Rings the bell on the terminal.

TERASE Erases the terminal screen.

TEXT * This subroutine outputs characters from a provided array to the screen, starting at the current alphanumeric cursor position. The text will be clipped at the edges of the screen. Assembly routine TOADE is used to unpack the text and convert it to ADE.

THOME This subroutine moves the graphics cursor to the home position, and switches the terminal to alphanumeric mode.

TINITT * This subroutine initializes the Graphics Package, clears the terminal screen, and positions the graphics cursor at (0,0) in the main viewing area. NCHMAX is set equal to 79 here.

TWAIT This subroutine causes the program to wait for a given number of seconds. This is accomplished in this version by the output of a string of non-interfering characters to the terminal.

WRTADE * This assembly routine converts an array of ADE characters to EBCDIC and sends them to the terminal with the interline characters suppressed. WRTADE is an entry point in LIBR5A.

RDADE * This assembly routine reads characters from the terminal, converts them from EBCDIC to ADE, and places them into a specified array. RDADE is an entry point in LIBR5A.

TOADE * This assembly routine converts a string of consecutive EBCDIC character into ADE characters and puts them into consecutive elements (right justified) of a full word array. TOADE is an entry point in LIBR5A.

TOEBCD * This assembly routine converts a string of consecutive ADE character into EBCDIC characters and puts them into consecutive elements (right justified) of a full word array. TOEBCD is an entry point in LIBR5A.

APPENDIX F:

THE IBM (CP/CMS) CHARACTER MANIPULATION PACKAGE SUBROUTINES (* indicates modified or added subroutines)

- DATEP * This subroutine calls IBM Operating System function DATIME for the date and converts it into 3A4 format.
- ENI * This subroutine encodes an integer into a floating point variable, left-justified and blank filled, and returns the floating point variable and the number of characters it contains.
- FREAD * This subroutine controls interactive I/O operations with the user via the terminal.
- FREAD2 * This subroutine controls interactive I/O operations with the user via the terminal or digitizing tablet.
- INCCHR * This subroutine takes a single character, left-justified and blank filled, and increments it by one.
- INCHAR * This logical function reads a line from the user's terminal or steering file as directed by subroutine FREAD.
- INCHR2 * This logical function reads a line from the user's terminal, steering file, or digitizing tablet as directed by subroutine FREAD2.

INCNM * This subroutine takes an alphanumeric eight character name and increments the numeric characters in the name by a specified amount. Calls assembly routine BTD.

INIHVL * This subroutine initializes the hardware value list for the computer hardware being used.

INMENU * This subroutine reads an input line from the tablet menu.

NUMCHR * This subroutine counts the number of non-blank characters in a single alphanumeric name.

PAKAL * This subroutine packs alphanumerical information into a single word in A4 format. Makes a call to assembly routine ENCODE for this.

PERCNT * This subroutine encodes an integer followed by a '%' sign into a floating point variable and then passes it to subroutine TEXT of the Graphics Package for inclusion in a plot. A call is made to assembly routines DECODE and ENCODE to aid in the process.

SECOND * This subroutine returns the elapsed CPU time in seconds. CPU time is obtained via a call to the system routine GETIME.

TIMEP * This subroutine gets the current time of day from the operating system, via a call to DATIME, and reformats it to 3A4 format.

UPKAL * This subroutine unpacks text in A4 format into a four word array with one character per word, right justified and zero (null) filled. UPKAL calls assembly routine DECOD for this.

DECODE * This added assembly routine unpacks a string of characters (4 per word) into a four word array with each word containing one left justified character. DECODE is an entry point in LIBR5A.

ENCODE * This added assembly routine packs a string of characters contained in single character words, into a single word. ENCODE is an entry point in LIBR5A.

DECOD * This added assembly routine unpacks the characters (maximum of 4) contained in a single word into four single character words, with the characters right justified and the word padded with zeroes (nulls). DECOD is an entry point in LIBR5A.

BTD * This added assembly routine transforms an integer into a string of alphanumeric characters. The string is right justified with the rest of the string filled with a character provided by the user. It additionally returns the number of characters in the string. BTD is an entry point in LIBR5A.

ENF * This assembly routine replaces a FORTRAN routine.
 It translates an array in 3A4 format into the
 floating point format 1PE10.3. ENF is an entry
 point in LIBR5A.

APPENDIX G:

THE IBM (CP/CMS DATA BASE MANAGEMENT PACKAGE SUBROUTINES
(* indicates a modified or added subroutine)

- CLOSEF * This subroutine closes an active file in the CMS environment via the CMS command FINIS. In addition, if the file being closed is GIFTS5 ESX, this subroutine calls for the deletion of GIFTS5 EST.
- CLOSLP * This subroutine spools to the line printer, if directed by the user, the line printer file via the CMS command PRINT.
- DEFIN * This subroutine associates direct access file identifiers with their respective logical unit numbers via the CMS command FILEDEF. It will also zero, or expand, a direct access UDB file as directed.
- DELETE * This subroutine makes direct access UDB files into 'dummy' files and deletes sequential UDB files via the CMS command ERASE.
- INITIO * This routine initializes the logical unit numbers for the direct access UDB files, defines all direct access UDB files, turns on the IBM CPU timing via a call to routine SETIME, and the current GIFTS version number. Additionally, if file GIFTS5 ESX exists, it is renamed GIFTS5 EST.

INRA This subroutine performs all input from the direct access UDB files.

OPENIF * This subroutine opens a specified sequential file for input by associating the file identifier with its respective logical unit number via the CMS command FILEDEF. If the file is GIFTS5 INF or GIFTS5 EST and does not exist on the user's disk space, it is specified as being on the GIFTS system's disk space.

OPENLP * This subroutine sets the line printer file logical unit number and associates the file identifier, provided by the user, with this logical unit number via the CMS command FILEDEF.

OPENOF * This subroutine opens a specified sequential file for output by associating the file identifier with its respective logical unit number via the CMS command FILEDEF. If the file to be opened is GIFTS5 EST, file GIFTS5 ESX is opened in its place.

OUTRA This subroutine performs all output to the direct access UDB files.

PRESNT * This logical function checks for the presence of a UDB file. If the file is sequential and not present, PRESNT is set equal to false, and if it is present, PRESNT is set equal to true. If the file is direct access, present on the user's disk

space, and a 'dummy' file, PRESNT is set equal to false. And if on the user's disk space and not a 'dummy' file, PRESNT is set equal to true. If the file is direct access and is not present on the user's disk space, program module execution is terminated.

RENAME * This subroutine uses the CMS command RENAME to effect the exchange of names between two UDB files. When renaming occurs, it also sets a switch to indicate to subroutine MANAGE that the renaming has taken place.

MANAGE * This subroutine will assign a logical unit number, if provided with the UDB file type, or the UDB file type, if provided with the logical unit number. If a file has been renamed, MANAGE will relate the old logical unit number with the new file type.

SLTASN * For this version, SLTASN is a dummy routine.

FRESLT * This subroutine was moved from LIBR1 for this version and is a dummy routine.

FRTCMX * This assembly routine allows CMS commands to be invoked from the executing program module. FRTCMX is an entry point in LIBR5A.

CONVRT * An assembly routine used to convert a four byte integer logical unit number into an eight byte real number. It is used in passing information on the logical unit number to FRTCMX for the execution of CMS commands requiring the number. CONVRT is an entry point in LIBR5A.


```
LIBR5 IS A COLLECTION OF ROUTINES FOR USE WITH THE GIFTS
LIBRARY LIBR5

ENTRY POINTS ARE: WRTADE
                   RDADE
                   TOADE
                   TCEBCD
                   DECODE
                   ENCODE
                   DECOD
                   BITD
                   ENF
                   FRTCMX
                   CONVRT

MACRO BEGIN
ENTRY USING &NAME
SAVE *15
L DROP (14,12),T,*
USING 12,ALIBR5A
LA 15,LIBR5A,12
ST 15,SAVEAREA
MEND 13,4(,15)
EJECT 15,8(,13)
START 0
USING *,12

LIBR5A
*
```



```

** START ROUTINES FOR GRAPHICS PACKAGE OF LIBR5.
**
** *****
** WRTADE CCNVERTS AN ARRAY OF ADE CHARACTERS TO EBCDIC AND SENDS THE
** CHARACTERS TO THE TERMINAL WITH INTERLINE CHARACTERS SUPPRESSED.
** *****
**
** FORTRAN CALLING SEQUENCE: CALL WRTADE(NCHAR,IARRAY)
**
** *****
** WHERE NCHAR = LENGTH OF IARRAY.
** IARRAY = INTEGER ARRAY, EACH ELEMENT CONTAINS
** AN ADE CHARACTER RIGHT JUSTIFIED.
** *****
**
** *****
** ENTRY WRTADE
** B 12(15)
** DC AL1(6)
** DC CL6,WRTADE
** SAVE (14,12)
** LRING R12,R15
** USING WRTADE,R12
** ST R13,SAVAR+4
** LA R13,SAVAR
**
** *****
**
** *****
** R2=ADDR OF PARM1
** R4=PARM1
** R4 > 0?
** BR IF POSITIVE
**
** *****
**
** *****
** R5=ADDR OF PARM2
** R6=ADDR OF OUTPUT BUFFER
** MOVE BYTE TO OUTPUT BUFFER
** NEXT WORD ON INPUT ARRAY
** NEXT BYTE IN OUTPUT STRING
**
** *****
**
** *****
** R4=PARM1
** R6=ADDR OF BUFF
** NUMBER OF CHARS TO TRANSLATE
** O(1,R6),TABLEOUT
** WRTERM BUFF,(R4),EDIT=LONG,COLOR=B
** WAITT
**
** *****
**
** *****
** R13,SAVAR+4
** LM R14,R12,12(R13)
** LA R15,0
** BR R14
** EJECT
** END WRTADE
**

```

```

LIB00370
LIB00380
LIB00390
LIB00400
LIB00410
LIB00420
LIB00430
LIB00440
LIB00450
LIB00460
LIB00470
LIB00480
LIB00490
LIB00500
LIB00510
LIB00520
LIB00530
LIB00540
LIB00550
LIB00560
LIB00570
LIB00580
LIB00590
LIB00600
LIB00610
LIB00620
LIB00630
LIB00640
LIB00650
LIB00660
LIB00670
LIB00680
LIB00690
LIB00700
LIB00710
LIB00720
LIB00730
LIB00740
LIB00750
LIB00760
LIB00770
LIB00780
LIB00790
LIB00800
LIB00810
LIB00820
LIB00830

```



```

*****
**      RDADE ACCEPTS CHARACTERS FROM THE TERMINAL, TRANSLATES THEM FROM
**      EBCDIC TO ADE, PUTTING ONE CHARACTER (RIGHT JUSTIFIED) IN EACH
**      ELEMENT OF IARAY.  NCHAR IS THEN ASSIGNED THE NUMBER OF
**      CHARACTERS READ.
**
**      FORTRAN CALLING SEQUENCE:  CALL RDADE(NCHAR,IARAY)
**
**      WHERE NCHAR = NUMBER OF CHARACTERS READ.
**      IARAY = INTEGER ARRAY, EACH ELEMENT WILL CONTAIN
**              AN ADE CHARACTER RIGHT JUSTIFIED.
*****
ENTRY RDADE
    B 10(15)
    DC AL1(5)
    DC CL5,RDADE
    SAVE (14,12)
    LR R12,R15
    USING RDADE,R12
    ST R13,SAVAR+4
    LA R13,SAVAR
    LR R2,R1
    RCTERM BUFF,EDIT=NO
    ST R0,LN
    L R4,0(R2)
    L R6,4(R2)
    L R8,R8
    IC R8,LN+3
    ST R8,0(R4)
    LTR R8,R8
    BNP RETIN
    STC R8,TRANIN+1
    LA R7,BUFF
    SR R9,R9
    TR 0(1,R7),TABLEIN
    ST R9,0(R6)
    MVC 3(1,R6),0(R7)
    LA R6,4(R6)
    LA R7,1(R7)
    BCT R8,INLOOP
    LM R13,SAVAR+4
    LA R14,R12,12(R13)
    LA R15,0
    BR R14
    EJECT
    END RDADE
*****
RDADE
*****
TMACIN
*****
R2=ADDR OF PARM LIST
*****
R4=ADDRESS OF NCHAR
R6=ADDRESS OF IARAY
*****
R8=LENGTH OF INPUT LINE
PUT LENGTH IN NCHAR
LN NOT POS?
*****
NUMBER OF CHARS TO BE TRANSLATED
R7=ADDR CF BUFF
R9=00000000
*****
ZERO OUTPUT WORD
MOVE FROM BUFF TO IARAY
NEXT WORD IN IARAY
NEXT BYTE IN BUFF
LOOP FOR NCHAR BYTES
*****
TRANIN
INLOOP
MOVIN
*****
RETIN
*****
*

```

```

LI800850
LI800860
LI800870
LI800880
LI800890
LI800900
LI800910
LI800920
LI800930
LI800940
LI800950
LI800960
LI800970
LI800980
LI800990
LI801000
LI801010
LI801020
LI801030
LI801040
LI801050
LI801060
LI801070
LI801080
LI801090
LI801100
LI801110
LI801120
LI801130
LI801140
LI801150
LI801160
LI801170
LI801180
LI801190
LI801200
LI801210
LI801220
LI801230
LI801240
LI801250
LI801270
LI801280
LI801290
LI801300
LI801310
LI801320

```



```

* ** ** ** ** ** ** ** ** ** **  LIB01330
* ** ** **  LIB01340
* ** ** **  LIB01350
* ** ** **  LIB01360
* ** ** **  LIB01370
* ** ** **  LIB01380
* ** ** **  LIB01390
* ** ** **  LIB01400
* ** ** **  LIB01410
* ** ** **  LIB01420
* ** ** **  LIB01430
* ** ** **  LIB01440
* ** ** **  LIB01450
* ** ** **  LIB01460
* ** ** **  LIB01470
* ** ** **  LIB01480
* ** ** **  LIB01490
* ** ** **  LIB01500
* ** ** **  LIB01510
* ** ** **  LIB01520
* ** ** **  LIB01530
* ** ** **  LIB01540
* ** ** **  LIB01550
* ** ** **  LIB01560
* ** ** **  LIB01570
* ** ** **  LIB01580
* ** ** **  LIB01590
* ** ** **  LIB01600
* ** ** **  LIB01610
* ** ** **  LIB01620
* ** ** **  LIB01630
* ** ** **  LIB01640
* ** ** **  LIB01650
* ** ** **  LIB01660
* ** ** **  LIB01670
* ** ** **  LIB01680
* ** ** **  LIB01690
* ** ** **  LIB01700
* ** ** **  LIB01710
* ** ** **  LIB01720

** ** ** **  *****
** ** **  SUBROUTINE TOADE CCONVERTS A STRING OF CONSECUTIVE EBCDIC CHARACTERS
** ** **  INTO ASCII DECIMAL EQUIVALENT CHARACTERS AND PUTS THEM INTO
** ** **  CONSECUTIVE ELEMENTS (RIGHT JUSTIFIED) OF A FULL WORD INTEGER ARRAY.
** ** **  *****
** ** **  FORTRAN CALLING SEQUENCE: CALL TOADE(NCHAR,KAM,KADE)
** ** **  *****
** ** **  WHERE: NCHAR= NUMBER OF BYTES IN INPUT STRING.
** ** **  KAM= INPUT STRING OF EBCDIC CHARACTERS.
** ** **  KADE= FULL WORD OUTPUT ARRAY OF LENGTH NCHAR.
** ** **  *****
** ** **  ENTRY TOADE
** ** **  B 12(0,15)
** ** **  DC AL1(6)
** ** **  DC CL6,TOADE*
** ** **  USING TOADE,R12
** ** **  SAVE (14,12),
** ** **  LR R12,R15
** ** **  ST R13,SAVAR+4
** ** **  LA R13,SAVAR
** ** **  *****
** ** **  L R2,0(R1)
** ** **  L R4,0(R2)
** ** **  L R5,4(R1)
** ** **  L R6,8(R1)
** ** **  L R8,ZEROES
** ** **  L R8,0(R6)
** ** **  ST MVC
** ** **  TR 3(1,R6),0(R5)
** ** **  LA R5,1(R5)
** ** **  LA R6,4(R6)
** ** **  BCT R4,LAM2AS
** ** **  *****
** ** **  L R13,SAVAR+4
** ** **  LM 14,12,12(13)
** ** **  LA R15,0
** ** **  BR 14
** ** **  EJECT
** ** **  END TOADE
** ** **  *****

```



```

**SUBROUTINE TOEBCD CONVERTS AN ADE CHARACTER RIGHT JUSTIFIED IN EACH
**ELEMENT OF THE INPUT ARRAY TO AN EBCDIC CHARACTER AND PUTS
**IT IN THE OUTPUT CHARACTER STRING.
**
**FORTRAN CALLING SEQUENCE: CALL TCEBCD(NCHAR,KADE,KAM)
**
**WHERE: NCHAR= NUMBER OF ELEMENTS IN INPUT ARRAY.
**        KADE= INPUT ARRAY OF NCHAR ELEMENTS, EACH ELEMENT
**              CONTAINS AN ADE CHARACTER RIGHT JUSTIFIED.
**              OUTPUT STRING OF NCHAR ADE CHARACTERS.
**
**ENTRY TOEBCD
**      B 12(0,15)
**      DC AL1(6)
**      DC CL6,TOEBCD'
**      SAVE (14,12)
**      LR R12,R15
**      USING TOEBCD,R12
**      ST R13,SAVAR+4
**      LA R13,SAVAR
**
**      R2=0(R1)
**      R4=0(R2)
**      R5=4(R1)
**      R6=8(R1)
**      MVC 0(1,R6),3(R5)
**      LA R5,4(R5)
**      R6,1(R6)
**      BCT R4,LAS2AM
**      R4,0(R2)
**      BCTR R4,0
**      STC R4,TRANS2+1
**      R6,8(R1)
**      L R6,0(1,R6),TABLECUT
**
**      R2=ADDR OF PARM1
**      R4=PARM1 OF PARM2
**      R5=ADDR OF PARM3
**      R6=ADDR OF PARM3
**      MOVE BYTE TC OUTPUT STRING
**      NEXT WORD ON INPUT ARRAY
**      NEXT WORD IN OUTPUT STRING
**      LOOP UNTIL NCHAR CHARACTERS ARE PROCESSED
**      R4=PARM1
**      DECREASE R4 BY 1
**      LENGTH OF STRING
**      R6=ADDR OF PARM3
**      R6=ADDR OF PARM3
**
**      R13,SAVAR+4
**      R4,12,12(13)
**      R15,0
**      BRCT 14
**      EJECT
**      ENC TOEBCD
**
**
**

```


THIS TABLE IS USED TO CONVERT ASCII TO EBCDIC

LIB02180
LIB02190
LIB02200
LIB02210
LIB02220
LIB02230
LIB02240
LIB02250
LIB02260
LIB02270
LIB02280
LIB02290
LIB02300
LIB02310
LIB02330
LIB02340
LIB02350
LIB02360
LIB02370
LIB02380
LIB02400
LIB02410
LIB02420
LIB02430
LIB02440
LIB02450
LIB02460
LIB02470
LIB02480
LIB02490
LIB02500
LIB02510
LIB02520
LIB02530
LIB02540
LIB02550
LIB02560
LIB02570
LIB02580
LIB02610
LIB02620
LIB02630
LIB02640
LIB02650
LIB02660
LIB02670

*

TABLEOUT EQU

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DC	X'00010203372D2E2F1605250B0C0D0E0F'	0														
DC	X'101112133C3D3E3F405A7F7B5B6C507D4D5D5C4E6B604861'	1														
DC	X'405A7F7B5B6C507D4D5D5C4E6B604861'	2														
DC	X'F0F1F2F3F4F5F6F7F8F97A5E4C7E6E6F'	3														
DC	X'7CC1C2C3C4C5C6C7C8C9D1D2D3D4D5D6'	4														
DC	X'D7D8D9E2E3E4E5E6E7E8E9ADE08D5F6D'	5														
DC	X'79818283848586878889919293949596'	6														
DC	X'979899A2A3A4A5A6A7A8A9CA06AD0A107'	7														

THIS TABLE IS USED TO CONVERT EBCDIC TO ASCII

LIB02180
LIB02190
LIB02200
LIB02210
LIB02220
LIB02230
LIB02240
LIB02250
LIB02260
LIB02270
LIB02280
LIB02290
LIB02300
LIB02310
LIB02330
LIB02340
LIB02350
LIB02360
LIB02370
LIB02380
LIB02400
LIB02410
LIB02420
LIB02430
LIB02440
LIB02450
LIB02460
LIB02470
LIB02480
LIB02490
LIB02500
LIB02510
LIB02520
LIB02530
LIB02540
LIB02550
LIB02560
LIB02570
LIB02580
LIB02610
LIB02620
LIB02630
LIB02640
LIB02650
LIB02660
LIB02670

*

TABLEIN EQU

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
DC	X'0001020300050C7F0000000080C0D0E0F'	0														
DC	X'101112133C000800181900001C1D1E1F'	1														
DC	X'00000000000A171B0000000000050607'	2														
DC	X'0000160000000040000000001415001A'	3														
DC	X'20000000000000000000002E3C282B00'	4														
DC	X'260000000000000000000021242A293B5E'	5														
DC	X'2D2F00000000000000007C2C255F3E3F'	6														
DC	X'000000000000000000000603A2340273D22'	7														
DC	X'00616263646566676869077B00000000'	8														
DC	X'006A6B6C6D6E6F707172007D00000000'	9														
DC	X'007E737475767778797A000000580000'	A														
DC	X'000000000000000000000000005D0000'	B														
DC	X'7B414243444546474849000000000000'	C														
DC	X'7D4A4B4C4D4E4F505152000000000000'	D														
DC	X'5C00535455565758595A000000000000'	E														
DC	X'30313233343536373839000000000000'	F														

*

REGEQU

DS 1F
DS 18F
DC X'00000000'
DC CL130'

END ROUTINES FOR GRAPHICS PACKAGE OF LIBR5.

```

*****PURPOSE.*****
*****THIS FUNTION PACKS A STRING OF N CHARACTERS*****
*****LEFT JUSTIFIED WITH THE REST BLANKED) IN*****
*****A STRING OF CHARACTERS (4 PER WORD)*****
*****CALL ENCODE(N,BUFF,LIST)*****
*****ARGUMENTS*****
*****N*****
*****BUFF*****
*****LIST*****
*****NUMBER OF CHARACTERS*****
*****ARRAY CONTAINING THE PACKED STRING*****
***** (4 CHARACTERS PER WORD)*****
***** ARRAY CONTAINING THE ORIGINAL STRING*****
***** (1 CHARACTER PER WORD)*****
*****R2-R4 = ADR. OF ARGUMENTS*****
*****BEGIN*****
*****LM*****
*****LR*****
*****BCTR*****
*****SRL*****
*****SLL*****
*****L*****
*****ST*****
*****IC*****
*****STC*****
*****LA*****
*****LAT*****
*****BCI TURN*****
*****REJECT*****
*****END ENCODE*****
*****ENCODE*****
*****2,5,0(1)*****
*****2,0(,2)*****
*****5,2*****
*****5,0*****
*****5,2*****
*****5,2*****
*****0,=CL4,*****
*****0,0(5,3)*****
*****6,0(,4)*****
*****6,0(,3)*****
*****4,4(,4)*****
*****4,1(,3)*****
*****2,ENCOD10*****
***** (14,12),T*****
*****BLANK THE LAST WORD OF BUF*****
*****LI B03040*****
*****LI B03050*****
*****LI B03060*****
*****LI B03070*****
*****LI B03080*****
*****LI B03090*****
*****LI B03100*****
*****LI B03110*****
*****LI B03120*****
*****LI B03130*****
*****LI B03140*****
*****LI B03150*****
*****LI B03160*****
*****LI B03170*****
*****LI B03180*****
*****LI B03190*****
*****LI B03200*****
*****LI B03210*****
*****LI B03220*****
*****LI B03230*****
*****LI B03240*****
*****LI B03250*****
*****LI B03260*****
*****LI B03270*****
*****LI B03280*****
*****LI B03290*****
*****LI B03300*****
*****LI B03310*****
*****LI B03320*****
*****LI B03330*****
*****LI B03340*****
*****LI B03350*****
*****LI B03360*****
*****LI B03370*****
*****LI B03380*****
*****LI B03390*****

```


B	ST	4,TEST	LI804250
	TM	TEST,X'0F'	LI804260
	BM	S	LI804270
	ST	5,UN	LI804280
	TM	TEST+1,X'0F'	LI804290
	BM	S	LI804300
	ST	5,UN	LI804310
	TM	TEST+2,X'0F'	LI804320
	BM	S	LI804330
	ST	5,UN	LI804340
	B	S	LI804350
* DBYTE	L	4,RES+4	LI804360
	S	5,QUATRE	LI804370
	CL	4,ZERO	LI804380
	BE	TBYTE	LI804390
	ST	4,TEST	LI804400
	B	B	LI804410
			LI804420
* TBYTE	L	4,RES+8	LI804430
	S	5,QUATRE	LI804440
	CL	4,ZERO	LI804450
	BE	VALNUL	LI804460
	ST	4,TEST	LI804470
	B	B	LI804480
			LI804490
* VALNUL	SR	5,5	LI804500
	B	PCSTITIF	LI804510
			LI804520
* S	TM	RES+11,X'10'	LI804530
	BZ	POSITIF	LI804540
	LA	6,RES	LI804550
	A	6,ONZE	LI804560
	SR	6,5	LI804570
	MVC	0(1,6),MOINS	LI804580
	A	5,UN	LI804590
			LI804600
* POSITIF	MVZ	RES+11(1),CF	LI804610
*			LI804620
	L	9,16(1)	LI804630
	LA	7,RES	LI804640
	LR	8,5	LI804650
BOUC	MVC	0(1,7),0(9)	LI804660
	A	7,UN	LI804670
	A	8,UN	LI804680
	C	8,DOUZE	LI804690
	BNE	BOUC	LI804700
			LI804710
*			LI804720

LI B04730
 LI B04740
 LI B04750
 LI B04760
 LI B04770
 LI B04780
 LI B04790
 LI B04800
 LI B04810
 LI B04820
 LI B04830
 LI B04840
 LI B04850
 LI B04860
 LI B04870
 LI B04880
 LI B04890
 LI B04900
 LI B04910
 LI B04920
 LI B04930
 LI B04940
 LI B04950
 LI B04960
 LI B04970
 LI B04980
 LI B04990
 LI B05000
 LI B05010
 LI B05020
 LI B05030
 LI B05040

```

7,9,4(1)
5,0(9)
10,0(8)
7,10
7,UN
11,RES+11
12,UN

* BOUCLE
MVC 0(17),0(11)
SR 10,12
SR 7,12
SR 11,12
C 10,NUL
BNE BOUCLE

*
LM 2,12,28(13)
MVI 12(13),X'FF'
BCR 15,14

* DBL
RES CL12
ZERO X'F0F0F0F0'
UN X'00000001'
QUATRE X'00000004'
ONZE X'0000000B'
DOUZE X'0000000C'
TEST CL4
NUL X'00000000'
MOINS X'60'
CF X'F0'

*
EJECT
END BTD

```



```

LIB05390
LIB05400
LIB05410
LIB05420
LIB05430
LIB05440
LIB05450
LIB05460
LIB05470
LIB05480
LIB05490
LIB05500
LIB05510
LIB05520
LIB05530
LIB05540
LIB05550
LIB05560
LIB05570
LIB05580
LIB05590
LIB05600
LIB05610
LIB05620
LIB05630
LIB05640
LIB05650
LIB05660
LIB05670
LIB05680
LIB05690
LIB05700
LIB05710
LIB05720
LIB05730
LIB05740
LIB05750
LIB05760
LIB05770
LIB05780
LIB05790
LIB05800
LIB05810
LIB05820
LIB05830
LIB05840
LIB05850
LIB05860

*****
START ROUTINES FOR DATA MANAGEMENT PACKAGE OF LIBR5.
*****
SUBROUTINE FRTCMX
*****
PURPOSE
TO ALLOW A USER TO INVOKE ALMOST ANY CP OR CMS COMMAND FROM
WITHIN A FCRTTRAN PROGRAM. NORMALLY AFTER THE INVOKED COM-
MAND IS EXECUTED, CONTROL RETURNS TO THE FCRTTRAN PROGRAM.
*****
USAGE CALL FRTCMX (CMSRC,TOKEN1,TOKEN2,...,TOKENL)
*****
DESCRIPTION OF PARAMETERS
*****
CMSRC IS THE CMS COMMAND RETURN CODE. IT CAN BE USED IN THE
FORTRAN CODE TO CHECK FOR ERRORS IN THE CMS COMMAND EXECUTION.
*****
EACH TOKEN IS A REAL*8 WORD CONTAINING LITERAL (HOLLERITH)
INFORMATION WHICH IS AN ELEMENT OF A CP OR CMS COMMAND WHICH
COULD OTHERWISE BE ISSUED BY THE USER AT A TERMINAL OR WITHIN
AN EXEC FILE.
*****
EACH TOKEN MUST BE PADDED WITH BLANKS SO THAT IT HAS EXACTLY
EIGHT CHARACTERS. THE INFORMATION IN THE TOKEN MUST BE LEFT-
JUSTIFIED.
*****
SEE EXAMPLES BELOW.
*****
REMARKS
1. CMSRC IS A 5 DIGIT INTEGER.
2. A MAXIMUM OF 20 TOKENS IS ALLCWD. IF MORE ARE ATTEMPTED
THE EFFECTIVE COMMAND IS TRUNCATED AFTER THE 20TH.
3. SUBJECT TO THE MAXIMUM, ANY NUMBER OF TOKENS IS ALLOWED.
4. IN GENERAL A COMMAND THAT RELGADS VIRTUAL CORE WILL NOT
BE ALLOWED. EXAMPLES: FORTHX, SCR IPT, XEDIT, ANY LANGUAGE
PROCESSOR, SORT, AND OTHER LARGE SYSTEM MODULES, SUCH AS
COPYFILE AND MOVEFILE. IT MAY BE POSSIBLE TO LOAD AND
EXECUTE A SMALL PROGRAM BY USING THE INCLUDE COMMAND.
5. THE FIRST TOKEN OF THE CP COMMAND MUST BE 'CP
WHEN AN EXEC FILE IS INVOKED, THE FIRST TOKEN MUST BE

```



```

** ** ** ** **
*EXEC      .
5.  AS CF NOV '81 YOU MUST NOT ABBREVIATE 'PRINTER' WHEN
    USING FRTCMX TO ISSUE FILE DEFINITIONS.

EXAMPLES
1.  TO CLEAR THE SCREEN OF A 3278 TERMINAL:
    CALL FRTCMX(IERR,'CLRSCRN')
    NOTE: CLRSCRN IS A BUILT-IN IPF MODULE.

2.  TO INVCKE AN EXEC FILE NAMED 'MYEXEC' WITH ARGUMENTS 'ARG1'
    AND 'ARG2'.
    CALL FRTCMX(IERR,'EXEC      ','MYEXEC      ','ARG1      ','ARG2

3.  TO ISSUE A CP SPOOL COMMAND TO CHANGE SECOND LABEL ON
    PRINTED OUTPUT:
    CALL FRTCMX(IERR,'CP      ','SPOOL      ','PRINTER      ','DIST
1    'NEWLABEL')

4.  TO ISSUE A FILEDEF TO SEND OUTPUT ON UNIT 6 TO PRINTER:
1    REAL*8 CCM(8),'FILEDEF      ','6      ','PRINTER      ','
1    'RECFM      ','FA      ','BLOCK      ','133
    CALL FRTCMX(COM(1),COM(2),COM(3),COM(4),COM(5),COM(6),
1    COM(7),COM(8))

    NOTE: THE LEFT PARENTHESIS WHICH DELIMITS OPTIONS MUST
    APPEAR AS A SEPARATE TOKEN. THE RIGHT PARENTHESIS MAY
    BE OMITTED.

IDENTIFICATION
PROGRAMMER:  ROGER HILLEARY      DATE:  APRIL 1981
WRITEUP REVISED NOV '81
MODIFIED TO RETURN CMS RETURN CODE BY:  HANS DOELMAN, JAN 82.

*****
FRTCMX      ENTRY  FRTCMX
            BC      15,8+4(15)      STANDARD FORTRAN LINKAGE
            DC      X'7'
            DC      CL7,FRTCMX
            STM      14,12,12(13)

            BALR     2,0
            USING    *,2
            *****
            L        6,0(1)
            ST       6,RCADR

*****
LIB05870
LIB05880
LIB05890
LIB05900
LIB05910
LIB05920
LIB05930
LIB05940
LIB05950
LIB05960
LIB05970
LIB05980
LIB05990
LIB06000
LIB06010
LIB06020
LIB06030
LIB06040
LIB06050
LIB06060
LIB06070
LIB06080
LIB06090
LIB06100
LIB06110
LIB06120
LIB06130
LIB06140
LIB06150
LIB06160
LIB06170
LIB06180
LIB06190
LIB06200
LIB06210
LIB06220
LIB06230
LIB06240
LIB06250
LIB06260
LIB06270
LIB06280
LIB06290
LIB06300
LIB06310
LIB06320
LIB06330
LIB06340

```



```

*****
* LA 1,4(1,0)*****
*
* LA 4,8
* LA 5,152
*
* LA 3,0
*
* L1
* LTR 6,0(1)
* LD 6,6(0)
* LA 1,4(1,0)
* STD 0,TOKENS(3)
* BCXLE 3,4,L1
* BC 15,L4
* LA 3,8(3,0)
* LD 0,DELIM
* STD 0,TOKENS(3)
*
* L3
* L4
*
* LA 1,TOKENS
* SVC 202
* DC AL4(*+4)
*****
* ST 2,RCADR
* ST 15,0(2)
*****
*
* LM 2,12,28(13)
* MVI 12(13),X'FF'
* BCR 15,14
*****
* RCADR DS IF
*****
*
* TOKENS
* OD
* DC CL8
* DC CL8
* DC CL8
* DC CL8
* DC CL8
* DC CL8
* DC CL8
* DC CL8
* DC CL8

```

```

8 IN REG 4
8*19 IN REG 5
ZERO IN REG 3 TO START
ADDRESS OF EACH TOKEN TO REG 6
TEST FOR LAST TOKEN (MINUS)
GET EACH TOKEN INTO FP REG 0
INCREMENT REG 1 BY 4
STORE EACH TOKEN INTO TABLE BELOW
BRANCH IF THIS WAS LAST TOKEN
MAX OF 20 TOKENS ALLOWED
UNCONDITIONAL BRANCH
INCREMENT REG 3 BY 4
STORE DELIMITER
GIVE CONTROL TO CMS
STANDARD RETURN LINKAGE

```

```

LIB06350
LIB06360
LIB06370
LIB06380
LIB06390
LIB06400
LIB06410
LIB06420
LIB06430
LIB06440
LIB06450
LIB06460
LIB06470
LIB06480
LIB06490
LIB06500
LIB06510
LIB06520
LIB06530
LIB06540
LIB06550
LIB06560
LIB06570
LIB06580
LIB06590
LIB06600
LIB06610
LIB06620
LIB06630
LIB06640
LIB06650
LIB06660
LIB06670
LIB06680
LIB06690
LIB06700
LIB06710
LIB06720
LIB06730
LIB06740
LIB06750
LIB06760
LIB06770
LIB06780
LIB06790
LIB06800
LIB06810
LIB06820

```


LIB06830
LIB06840
LIB06850
LIB06860
LIB06870
LIB06880
LIB06890
LIB06900
LIB06910
LIB06920
LIB06930

.....

DC CL8'
DC CL8'
DC CL8'
DC CL8'
DC CL8'
DC CL8'
DC CL8'
DC 8X'FF'
EJECT
END FRTCMX

DELIM
*

87

APPENDIX I: SCME IBM INSTALLATION TOOLS

FILE TAPE2MVS JCL

```
//HUNDLEY1 JOB (3161,0020),'R. HUNDLEY SMC 2436',CLASS=D
//XX PROC
// EXEC PGM=IEBGENER
//SYSPRINT DD DUMMY
//SYSIN DD DUMMY
//SYSUT1 DD UNIT=3400-4,VOL=SFR=GIFTS2,
// DISP=(OLD,PASS),LABFL=(&FN,RLP,,IN),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=1600,DEN=2,OPTCD=Q)
//SYSUT2 DD UNIT=3350,VOL=SER=MVS004,
// DISP=(NEW,KEEP),SPACE=(CYL,(1,1)),
// DCB=(RECFM=FB,LRECL=80,BLKSIZE=6400),
// DSN=S3161.&FNAME
// PEND
// EXEC XX,FN=1,FNAME=FILE1
// EXEC XX,FN=2,FNAME=FILE2
// EXEC XX,FN=3,FNAME=FILE3
// EXEC XX,FN=4,FNAME=FILE4
// EXEC XX,FN=5,FNAME=FILE5
// EXEC XX,FN=6,FNAME=FILE6
// EXEC XX,FN=7,FNAME=FILE7
// EXEC XX,FN=8,FNAME=FILE8
// EXEC XX,FN=9,FNAME=FILE9
// EXEC XX,FN=10,FNAME=FILE10
// EXEC XX,FN=11,FNAME=FILE11
// EXEC XX,FN=12,FNAME=FILE12
// EXEC XX,FN=13,FNAME=FILE13
// EXEC XX,FN=14,FNAME=FILE14
// EXEC XX,FN=15,FNAME=FILE15
// EXEC XX,FN=16,FNAME=FILE16
// EXEC XX,FN=17,FNAME=FILE17
// EXEC XX,FN=18,FNAME=FILE18
// EXEC XX,FN=19,FNAME=FILE19
// EXEC XX,FN=20,FNAME=FILE20
// EXEC XX,FN=21,FNAME=FILE21
// EXEC XX,FN=22,FNAME=FILE22
// EXEC XX,FN=23,FNAME=FILE23
// EXEC XX,FN=24,FNAME=FILE24
// EXEC XX,FN=25,FNAME=FILE25
// EXEC XX,FN=26,FNAME=FILE26
// EXEC XX,FN=27,FNAME=FILE27
// EXEC XX,FN=28,FNAME=FILE28
// EXEC XX,FN=29,FNAME=FILE29
// EXEC XX,FN=30,FNAME=FILE30
// EXEC XX,FN=31,FNAME=FILE31
// EXEC XX,FN=32,FNAME=FILE32
// EXEC XX,FN=33,FNAME=FILE33
// EXEC XX,FN=34,FNAME=FILE34
// EXEC XX,FN=35,FNAME=FILE35
// EXEC XX,FN=36,FNAME=FILE36
// EXEC XX,FN=37,FNAME=FILE37
```


FILE MVS2CMS EXEC

```
*****
*
*      MVS2CMS EXEC
*
*      THIS EXEC WILL CAUSE THE TRANSFER OF THE GIFTS SOURCE
*      CODE FROM MVS DISK MVS004 TO THE USER'S CMS DISK SPACE.
*
*      PRIOR TO USE OF THIS EXEC, LINK TO MVS004 BY ISSUING:
*
*          CP LINK MVS 36B 36B RR
*          ACC 36B E
*
*****
FILEDEF INMOVE E DSN S3161 FILE1
FILEDEF CUTMOVE DISK GIFTS5 INF
MOVEFILE INMOVE CUTMCVE
FILEDEF INMOVE E DSN S3161 FILE2
FILEDEF OUTMOVE DISK AUTOL FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY AUTOL FORTRAN A = PACKED = (PA
ERASE AUTOL FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE3
FILEDEF OUTMOVE DISK BULKF FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY BULKF FORTRAN A = PACKED = (PA
ERASE BULKF FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE4
FILEDEF OUTMOVE DISK BULKLB FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY BULKLB FORTRAN A = PACKED = (PA
ERASE BULKLB FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE5
FILEDEF OUTMOVE DISK BULKM FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY BULKM FORTRAN A = PACKED = (PA
ERASE BULKM FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE6
FILEDEF OUTMOVE DISK BULKS FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY BULKS FORTRAN A = PACKED = (PA
ERASE BULKS FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE7
FILEDEF OUTMOVE DISK DECOM FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY DECOM FORTRAN A = PACKED = (PA
ERASE DECOM FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE8
FILEDEF OUTMOVE DISK DEFCS FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY DEFCS FORTRAN A = PACKED = (PA
ERASE DEFCS FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE9
FILEDEF OUTMOVE DISK DEFL FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY DEFL FORTRAN A = PACKED = (PA
ERASE DEFL FORTRAN A
```



```

FILEDEF INMOVE E DSN S3161 FILE10
FILEDEF OUTMOVE DISK EDITLB FCRTRAN
MOVEFILE INMCVE CUTMCVE
COPY EDITLB FORTRAN A = PACKED = (PA
ERASE EDITLB FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE11
FILEDEF OUTMCVE DISK EDITM FORTRAN
MOVEFILE INMCVE OUTMOVE
COPY EDITM FORTRAN A = PACKED = (PA
ERASE EDITM FCRTRAN A
FILEDEF INMOVE E DSN S3161 FILE12
FILEDEF OUTMOVE DISK EDITS FCRTRAN
MOVEFILE INMCVE OUTMOVE
COPY EDITS FORTRAN A = PACKED = (PA
ERASE EDITS FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE13
FILEDEF OUTMCVE DISK ESTIM FCRTRAN
MOVEFILE INMCVE OUTMOVE
COPY ESTIM FORTRAN A = PACKED = (PA
ERASE ESTIM FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE14
FILEDEF OUTMOVE DISK LOADS FORTRAN
MOVEFILE INMOVE OUTMCVE
COPY LCADS FORTRAN A = PACKED = (PA
ERASE LOADS FCRTRAN A
FILEDEF INMOVE E DSN S3161 FILE15
FILEDEF OUTMOVE DISK LOCAL FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY LCCAL FORTRAN A = PACKED = (PA
ERASE LOCAL FCRTRAN A
FILEDEF INMOVE E DSN S3161 FILE16
FILEDEF OUTMOVE DISK OPTIM FORTRAN
MOVEFILE INMOVE CUTMCVE
COPY CPTIM FORTRAN A = PACKED = (PA
ERASE OPTIM FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE17
FILEDEF OUTMOVE DISK PRINT FORTRAN
MOVEFILE INMCVE CUTMCVE
COPY PRINT FORTRAN A = PACKED = (PA
ERASE PRINT FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE18
FILEDEF OUTMOVE DISK REDCS FORTRAN
MOVEFILE INMCVE OUTMOVE
COPY REDCS FORTRAN A = PACKED = (PA
ERASE REDCS FCRTRAN A
FILEDEF INMCVE E DSN S3161 FILE19
FILEDEF OUTMOVE DISK RESIDU FCRTRAN
MOVEFILE INMCVE CUTMCVE
COPY RESIDU FORTRAN A = PACKED = (PA
ERASE RESIDU FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE20
FILEDEF OUTMOVE DISK RESULT FCRTRAN
MOVEFILE INMOVE OUTMCVE
COPY RESULT FORTRAN A = PACKED = (PA
ERASE RESULT FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE21
FILEDEF OUTMCVE DISK SAVEK FCRTRAN
MOVEFILE INMCVE OUTMOVE
COPY SAVEK FORTRAN A = PACKED = (PA
ERASE SAVEK FCRTRAN A
FILEDEF INMOVE E DSN S3161 FILE22
FILEDEF OUTMCVE DISK STIFF FORTRAN
MOVEFILE INMOVE OUTMCVE
COPY STIFF FORTRAN A = PACKED = (PA
ERASE STIFF FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE23
FILEDEF OUTMOVE DISK STRESS FORTRAN

```



```

MOVEFILE INMCVE OUTMOVE
COPY STRESS FORTRAN A = PACKED = (PA
ERASE STRESS FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE24
FILEDEF OUTMOVE DISK SUBS FORTRAN
MOVEFILE INMCVE OUTMCVE
COPY SUBS FORTRAN A = PACKED = (PA
ERASE SUBS FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE25
FILEDEF OUTMOVE DISK TRAN1 FORTRAN
MOVEFILE INMCVE OUTMCVE
COPY TRAN1 FORTRAN A = PACKED = (PA
ERASE TRAN1 FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE26
FILEDEF OUTMOVE DISK TRAN2 FORTRAN
MOVEFILE INMCVE OUTMCVE
COPY TRAN2 FORTRAN A = PACKED = (PA
ERASE TRAN2 FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE27
FILEDEF OUTMOVE DISK TRANS FORTRAN
MOVEFILE INMCVE OUTMCVE
COPY TRANS FORTRAN A = PACKED = (PA
ERASE TRANS FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE28
FILEDEF OUTMOVE DISK TESTCS FORTRAN
MOVEFILE INMCVE OUTMCVE
COPY TESTCS FORTRAN A = PACKED = (PA
ERASE TESTCS FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE29
FILEDEF OUTMOVE DISK TESTIC FORTRAN
MOVEFILE INMCVE OUTMCVE
COPY TESTIO FORTRAN A = PACKED = (PA
ERASE TESTIO FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE30
FILEDEF OUTMOVE DISK TEST FORTRAN
MOVEFILE INMCVE OUTMCVE
COPY TEST FORTRAN A = PACKED = (PA
ERASE TEST FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE31
FILEDEF OUTMCVE DISK TEST2 FORTRAN
MOVEFILE INMCVE OUTMCVE
COPY TEST2 FORTRAN A = PACKED = (PA
ERASE TEST2 FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE32
FILEDEF OUTMOVE DISK TSTPLT FORTRAN
MOVEFILE INMCVE OUTMCVE
COPY TSTPLT FORTRAN A = PACKED = (PA
ERASE TSTPLT FORTRAN A
FILEDEF INMOVE E DSN S3161 FILE33
FILEDEF OUTMOVE DISK LIBR1 FORTRAN
MOVEFILE INMCVE OUTMCVE
FILEDEF INMOVE E DSN S3161 FILE34
FILEDEF OUTMOVE DISK LIBR2 FORTRAN
MOVEFILE INMCVE OUTMCVE
FILEDEF INMOVE E DSN S3161 FILE35
FILEDEF OUTMOVE DISK LIBR3 FORTRAN
MOVEFILE INMCVE OUTMCVE
FILEDEF INMOVE E DSN S3161 FILE36
FILEDEF OUTMOVE DISK LIBR4 FORTRAN
MOVEFILE INMCVE OUTMCVE
FILEDEF INMOVE E DSN S3161 FILE37
FILEDEF OUTMOVE DISK LIBR5 FORTRAN
MOVEFILE INMCVE OUTMCVE
REL E (DET
FLIST

```


APPENDIX J:

GIFTS USER INSTRUCTIONS FOR THE IBM (CP/CMS) VERSION

TERMINAL USAGE

GIFTS may be executed on any terminal, but if the terminal is not a Tektronix 4000 series graphics terminal, plot commands must not be used as they will be meaningless and may cause termination of execution. If using a non-graphics terminal, the user will have only the alphanumeric output from GIFTS available.

GIFTS ANALYSIS PROCEDURES

The analysis procedures described in GIFTS User's Reference Manual are used, but the module IBMUDB must first be executed to create the data base for the analysis. Following the execution of IBMUDB, the user will have 48 (small) files on the 'A' disk. Upon the completion of an analysis, it is the responsibility of the user to erase these files, as needed, from his disk space.

STORAGE REQUIREMENTS

To ensure sufficient core for an analysis, prior to executing a module, issue the command:

CP DEFINE STORAGE 1M

which will place you in the CP operating environment. To return to the CMS environment, issue the command:

CP IPL CMS.

LINKING TO THE GIFTS SYSTEM'S DISK SPACE

In order to perform a GIFTS analysis, the GIFTS system's disk space must be linked to and accessed as the user's 'C' disk. This can be accomplished by issuing the command (user input in lower case, computer output in upper case):

```
cp link 316lp 191 199 rr
ENTER READ PASSWORD:
gifts
R; T=0.01/0.01 20:25:17
access 199 c
C (199) R/O
R; T=0.01/0.02 20:25:25
```

Once linked and accessed as the user's 'C' disk, the GIFTS system is ready for use.

ENTERING A BLANK LINE IN GIFTS

At times, during execution of the GIFTS modules, it will be necessary to issue a blank line, or GIFTS may even prompt the user to enter a carriage return. It is of the utmost importance in both cases, that a space, plus a carriage return be entered. If a carriage return alone is entered, program module execution will be terminated.

EXECUTING A GIFTS PROGRAM MODULE

To execute a GIFTS program module, issue the command:

RUN XXXXXX

where XXXXXX is the name of the desired module. This will cause the desired module and necessary libraries to be loaded and executed. The user is reminded that module IBMUDB must be executed at the start of each analysis.

LIST OF REFERENCES

1. University of Arizona, GIFTS User's Reference Manual, 1981.
2. University of Arizona, GIFTS System's Manual, 1979.
3. University of Arizona, GIFTS Installation Manual, 1981.
4. University of Arizona, GIFTS Theoretical Manual, 1979.
5. IBM Corporation, IBM Virtual Machine/System Product: CMS Command and Macro Reference, SC19-6209-0, 1st. Ed., 1980.
6. IBM Corporation, IBM Virtual Machine/System Product: CP Command Reference for General Users, SC19-6211-0, 1st. Ed., 1980.
7. IBM Corporation, IBM Virtual Machine/System Product: CMS User's Guide, SC19-6210-0, 1st. Ed., 1980.
8. IBM Corporation, IBM System/370 Principles of Operation, GA22-7000-6, 7th. Ed., 1980.
9. IBM Corporation, IBM System/360 and System/370 FORTRAN IV Language, GC-28-6515-10, 11th. Ed., 1974.
10. IBM Corporation, IBM System/370 Operating System FORTRAN IV (G and H) Programers Guide, GC28-6817-4, 5th. Ed., 1975.
11. IBM Corporation, IBM System Reference Library - OS Utilities, GC28-6585-15, 16th. Ed., 1973.
12. Newman, W. M. and Spronll, R. F., Principles of Interactive Computer Grahics, 2nd. Ed., McGraw-Hill, 1979.
13. Weik, M. H., Standard Dictionary of Computers and Information Processing, 1st. Ed., Hayden Books, 1969.
14. Zienkiewicz, O. C., The Finite Element Method, 3rd. Ed., McGraw-Hill, 1979.
15. IBM Corporation, IBM System/370 Reference Summary, GX20-1850-3, 4th. Ed., 1976.

INITIAL DISTRIBUTION LIST

	No. Copies
1. Defense Technical Information Center Cameron Station Alexandria, Virginia 22314	2
2. Library, Code 0142 Naval Postgraduate School Monterey, California 93940	2
3. Department Chairman, Code 69Mx Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93940	1
4. Professor Gilles Cantin, Code 69Ci Department of Mechanical Engineering Naval Postgraduate School Monterey, California 93940	5
5. Supervisor of Shipbuilding, Conversion and Repair, USN Attn: LT Ronnie Hundley, USN Long Beach Naval Shipyard Long Beach, California 90822	2
6. Professor Hussein A. Kamel University of Arizona College of Engineering Aerospace and Mechanical Engineering Dept. Interactive Graphics Engineering Laboratory AME Building, Room 2101 Tucson, Arizona 85721	2

Thesis

H9247

c.1

Hundley

27135
A version of the
graphics-oriented in-
teractive finite ele-
ment time-sharing
system (GIFTS) for an
IBM with CP/CMS.

Thesis

H9247

c.1

Hundley

27135
A version of the
graphics-oriented in-
teractive finite ele-
ment time-sharing
system (GIFTS) for an
IBM with CP/CMS.

A version of the graphics-oriented
3 2768 002 13259 9
DUDLEY KNOX LIBRARY